



Documentation: TextVision

AIM - Artificial Intelligence for Multimodal & Multisensor Applications

Members:

Melis Aslan, Maximilian Bosse,
Daniel Ehlers, Marlon Hinz,
Philipp Olschewski, Jannik Podszun,
Elias Scharlach, Leon Selzer, Yukun Wu

Submission: 2nd March 2025

Supervisors:

Prof. Dr.-Ing. Daniel Sonntag & Alikı Anagnostopoulou M. Sc.

Contents

List of tables	vi
List of illustrations	viii
1. Introduction	1
2. Fundamentals	3
2.1. Related Work	3
2.1.1. AI-Assisted Document Processing	3
2.1.2. Prompt Engineering and Optimization	3
2.1.3. Intelligent User Interfaces for LLMs	4
2.1.4. Positioning of TextVision	4
2.2. Foundations of Artificial Intelligence	5
2.2.1. Machine Learning	6
2.2.2. Neural Networks	8
2.2.3. Deep Learning	9
2.2.4. Generative AI & Natural Language Processing	10
2.2.5. Large Language Models and Multimodal Models	12
2.2.6. Prompt Engineering	16
2.2.7. Fine-Tuning	17
2.2.8. Introduction to Chatbots	18
2.2.9. Human Computer Interaction	19
2.2.10. Intelligent User Interfaces	20
2.2.11. Multimodality	21
2.3. Technical Fundamentals	22
2.3.1. Distributed applications - Client-Server Model	22
2.3.2. Hypertext Transfer Protocol – HTTP	24
2.3.3. RESTful Web Services	26
2.3.4. Websocket	29
2.3.5. JSON	31
2.3.6. JSON Web Token (JWT)	32
3. Project Concept	35
3.1. Project Description	35
3.2. Project Management and Tools	35
3.2.1. Tools	36
3.2.2. Team Organization and Workflow	36
3.3. Scenario-Based Exploration of Key Use Cases & Features	39
3.3.1. Reviewing and Summarizing Research Papers	41
3.3.2. Designing and optimizing research prompts	42
3.3.3. Cross-referencing and contextualizing information	43
3.4. User Interface	44

3.5.	Integration of LLMs & Prompting Concept	45
3.6.	System Architecture	46
3.6.1.	Frontend	47
3.6.2.	Backend	48
3.6.3.	AI-Backend	49
3.7.	Documentation JSDoc	49
4.	Interfaces (API)	53
4.1.	Data-exchange between Frontend and Backend	53
4.1.1.	Authentication API	53
4.1.2.	ProjectCenter API	54
4.1.3.	PDF Viewer API	56
4.1.4.	DocumentEditor API	57
4.1.5.	Chat API	58
4.1.6.	Multimodality API	60
4.1.7.	PromptDesigner API	61
4.2.	Data-exchange between Backend and AI-Backend	63
5.	Frontend	66
5.1.	Overview of the Frontend	66
5.1.1.	Folder Structure	66
5.1.2.	UI Layout	67
5.1.3.	Routing and Context	69
5.1.4.	Communication to Backend	71
5.2.	Authentication	72
5.2.1.	Register	72
5.2.2.	Login	74
5.2.3.	Logout	74
5.2.4.	Delete Account	74
5.2.5.	Technical specialities	75
5.3.	ProjectCenter	79
5.4.	PDF Viewer	81
5.5.	DocumentEditor	85
5.5.1.	Component Overview	85
5.5.2.	Functionality of the DocumentEditor	87
5.5.3.	Speech-to-Text and Text-to-Speech	88
5.5.4.	Exporting Documents as PDF	88
5.5.5.	Resetting the Editor	89
5.5.6.	Integration with Chat Functionality	89
5.6.	Chat	89
5.6.1.	Structure and Functionality	90
5.6.2.	Interaction with PDF Viewer and DocumentEditor	97
5.7.	Multimodality	99
5.7.1.	Speech-to-Text combined with the DocumentEditor	99

5.7.2.	Text-to-Speech combined with the DocumentEditor	101
5.7.3.	Voice Input combined with the Chat	102
5.7.4.	Voice Commands	104
5.8.	PromptDesigner	108
5.8.1.	Component Overview	108
5.8.2.	Functionality of the PromptDesigner	109
5.8.3.	Error Handling and User Feedback	112
5.9.	FAQ	112
5.9.1.	Memoization (<code>useMemo</code>) for Efficient Rendering	113
5.10.	Test	115
5.11.	Known Problems	117
6.	Backend	122
6.1.	Overview of the Backend Specification	122
6.1.1.	Configuration	122
6.1.2.	Project Structure	122
6.1.3.	Dependencies	123
6.2.	Database Integration	125
6.2.1.	Users	125
6.2.2.	Model	126
6.2.3.	PDF	127
6.2.4.	Workspaces	128
6.3.	Authentication and users	130
6.3.1.	BearerHandler	130
6.3.2.	CipherHelper	130
6.3.3.	AuthController	132
6.3.4.	AuthService	133
6.3.5.	UserService	133
6.4.	Backend Functionlity	133
6.4.1.	UserRepository	133
6.4.2.	PDF Viewer	134
6.4.3.	DocumentEditor	137
6.4.4.	Chat	138
6.4.5.	PromptDesigner	140
6.4.6.	Workspaces	143
6.4.7.	Tests	145
6.5.	Known Problems	146
7.	AI-Backend	148
7.1.	Description and Overview	148
7.1.1.	System Requirements and Development Stack	149
7.1.2.	Folder Structure	150
7.2.	Implementation	151
7.2.1.	Handlers	152

7.2.2.	Fine-Tuning	159
7.2.3.	Prompt management	165
7.2.4.	File Processing	171
7.3.	Tests	176
7.3.1.	Test Environment Structure	176
7.3.2.	Test Overview	176
7.3.3.	Test Execution	178
7.4.	Known Problems	179
7.4.1.	Performance Considerations	179
7.4.2.	Model Behavior and Limitations	179
7.4.3.	Token and Memory Management	180
7.4.4.	Document Processing Limitations	181
7.4.5.	Technical Implementation Issues	181
8.	Design & Procedure of the User Study	182
8.1.	Research Objectives and Hypotheses	182
8.2.	Planning of the User Study	183
8.2.1.	Recruitment of Participants and Grouping	183
8.2.2.	Questionnaire Selection	184
8.3.	Execution of the User Study	185
9.	Results of the User Study	187
9.1.	Results of the Basic Questionnaire	187
9.2.	Results of System Usability Scale	188
9.3.	Results of NASA-TLX	189
9.4.	Additional Findings	190
9.5.	Discussion of Interview Feedback & General Insights	190
9.5.1.	Issues and Bugs identified during the User Study	192
9.6.	Evaluation of the User Study	193
9.6.1.	Research Question	193
9.6.2.	Sub-Question	194
9.6.3.	Hypotheses Evaluation	195
9.6.4.	Design & Development Implications	195
9.7.	System Requirements for TextVision	195
10.	Conclusion	197
11.	Outlook: Development and Expansion	199
	Bibliography	201
A.	Appendix	ix
A.1.	User Study	ix
A.1.1.	Application for Ethical Approval	ix

A.1.2. Promotional Text	xvii
A.1.3. Basic Questionnaire	xix
A.1.4. System Usability Scale (SUS)	xxii
A.1.5. NASA-TLX	xxiv
A.1.6. Interview Questions	xxvi
A.1.7. Results: System Usability Scale (SUS)	xxx
A.1.8. Results: NASA-TLX	xxxii
A.1.9. Text for more Information	xxxiv
A.1.10. Informed Consent Form	xxxvii
A.1.11. Exemplary (Deepfake) Paper used in User Study	xli
A.2. Transcripts of semi-structured Interviews	xlvi
A.3. Troubleshooting and Logging	cxvii
A.4. Fine-tuning script	cxviii
A.5. Fine-tuning evaluation script	cxxvi

List of Tables

1.	Comparison HTTP and WebSocket (Qigang and Sun, 2012)	31
2.	Roles and Responsibilities of Team Members	37
3.	Endpoints of the Authentication API	53
4.	Endpoints of the Projectcenter API	55
5.	Endpoints of the PDF Viewer API	57
6.	Endpoints of the DocumentEditor API	58
7.	Endpoints of the Chat API	59
8.	Endpoints of the Multimodality API	60
9.	Endpoints of the PromptDesigner API	61
10.	Endpoints for communication between Backend and AI-Backend	63
11.	Simple UML-like class diagram for the <code>Workspace</code> class	80
12.	Table of Categories, Descriptions, and Example Questions	113
13.	Results Fine Tuning	164

List of Figures

1.	Basic concepts of AI according to Sonnet, 2022	5
2.	Supervised Learning according to Buxmann and Schmidt, 2021	7
3.	Unsupervised Learning according to Buxmann and Schmidt, 2021	7
4.	Reinforcement Learning according to Buxmann and Schmidt, 2021	8
5.	Simple perceptron according to Sonnet, 2022	8
6.	Artificial Neural Network according to Buxmann and Schmidt, 2021	10
7.	NLP process according to Jungmann et al., 2018	11
8.	Transformer Architecture (Vaswani et al., 2017)	13
9.	Tokenizing techniques according to Amaratunga, 2023	15
10.	Ways of Interaction (Oviatt et al., 2017)	20
11.	Connecting AI and HCI with IUI (Oviatt et al., 2017)	21
12.	Client-Server Model	23
13.	Interface	24
14.	HTTP-Request	25
15.	HTTP methods - CRUD	25
16.	Example JWT: Encoding/Decoding	33
17.	Authentication process	34
18.	Important Tools	36
19.	SCRUM in PG AIM	38
20.	Illustration of the most relevant interactions between TextVision's core components (PDF Viewer, AI Assistant, Document Editor, and PromptDesigner).	40
21.	Illustration of how a researcher would generally utilize TextVision to enhance their productivity.	41
22.	Illustration of the prompting process in TextVision.	42

23.	Illustration of cross-referencing and contextualization when working with multiple uploaded PDFs.	43
24.	Initial Mockup of TextVision	44
25.	System architecture overview of TextVision	47
26.	Home-Page of TextVision	68
27.	Frontend: Structure and Routing	70
28.	User Interface: Register	73
29.	User Interface: User Info	74
30.	User Interface: Confirmation Delete Account	75
31.	Browser local storage: authentication, persist and workspace	76
32.	HTTP-Interceptor: AxiosPrivate-Instance communication with the Backend	79
33.	GUI of the ProjectCenter	81
34.	PDF Viewer component	82
35.	PDF Viewer after document upload	82
36.	Simplified diagram of the PDF Viewer Component	83
37.	Diagram for PDF Annotation Handling	84
38.	User Interface: DocumentEditor	85
39.	UML: DocumentEditor Class	86
40.	Chat interface	90
41.	Simplified UML Diagram of the ChatComponent	91
42.	Sequence Diagram of the authenticated WebSocket connection	92
43.	Comparison of OpenAI(top) and TextVision(bottom) prompt suggestions	95
44.	State Diagram of the prompt suggestion lifecycle	96
45.	All possible message forms	96
46.	Simplified sequence diagram of PDF Viewer and Chat interaction	98
47.	User Interface: DocumentEditor	100
48.	Chat interface	103
49.	UML Activity Diagram: Voice Command Processing	106
50.	UML Component Diagram: Voice Command Component Interaction	108
51.	UML: PromptDesigner Class	109
52.	User Interface: PromptDesigner (Prompt Section)	110
53.	User Interface: PromptDesigner (Evaluation Section)	111
54.	User Interface: PromptDesigner (Optimization Section)	111
55.	Overview of the FAQ Page	112
56.	Contact Modal of the FAQ Page	113
57.	Duplication of the prompt suggestion	120
58.	Comparison of normal(left) and error(right) display of the highlight plugin from React-Pdf-Viewer	121
59.	Schema representation of the user collection	126
60.	Schema representation of the model collection	127
61.	Schema representation of the PDF collection	128
62.	Schema representation of the workspaces collection	131
63.	UML User	132

64.	UML PDF Helper	134
65.	UML Editor	137
66.	UML Chat	139
67.	Prompt API	141
68.	Workspace API	143
69.	Architecture of the TextVision Python AI backend	151
70.	Handler Implementation	153
71.	Request Processing Pipeline	157
72.	Prompt Management System Architecture	165
73.	Prompt Evaluation and Optimization Workflow	168
74.	Suggestion Generation Workflow	169
75.	File Processing System	171
76.	Vector Store Architecture showing component relationships	173
77.	Search Operations Workflow	174
78.	Scatter plot of varying sample sizes in user studies (Caine, 2016)	184
79.	Grouping based on Question 5 of the Basic Questionnaire	188
80.	Total Score of each Category	189

1. Introduction

The increasing adoption of large language models (LLMs) has led to significant advancements in natural language processing. However, the full potential of these models remains largely untapped due to limitations in existing user interfaces and interaction paradigms. One of the primary challenges in leveraging LLMs effectively lies in prompt engineering, which often requires substantial domain expertise and iterative refinement (Zamfirescu-Pereira et al., 2023). Many current applications rely only on text-based inputs and outputs, failing to accommodate the multimodal nature of real-world information processing tasks (Oviatt et al., 2017). This limitation is particularly pronounced in the analysis of complex document formats such as PDFs, where textual content is often connected with figures, tables, and equations.

The web-application TextVision aims to bridge this gap by integrating LLMs into an interactive, multimodal user interface that try to enhance the efficiency and accessibility of artificial intelligence (AI) document analysis. The system provides users with a dedicated project center that stores their prompts, chat history, and document modifications, thereby enabling an iterative workflow for refining and applying AI-generated outputs. This context-aware environment facilitates a structured approach to prompt engineering, allowing users to optimize their interactions with LLMs systematically before deploying them in research or professional settings.

One of the core innovations of TextVision is its integration of LLMs with an interactive PDF viewer and editor, which transforms traditional document processing workflows. Unlike existing tools that primarily focus on static text extraction, TextVision incorporates multimodal input modalities, including text, images, and voice commands, to provide a more intuitive and flexible interaction (Hewett et al., 1992). This approach aligns with contemporary research in human-computer interaction (HCI) and intelligent user interfaces (IUIs) (Oviatt et al., 2017), ensuring that AI assistance is seamlessly embedded into document analysis tasks.

A key shortcoming of current AI-assisted document processing tools is their lack of contextual awareness. Most existing solutions operate on generic prompts without adapting to the specific requirements of different disciplines. (Sučik et al., 2023) TextVision addresses this limitation by implementing an AI-assisted knowledge management component, which utilizes proprietary and open-source LLMs, including models fine-tuned for scientific research applications (Hu et al., 2021). By empowering users to create, store, and refine custom prompt templates, TextVision enhances adaptability and usability across various domains, ensuring that AI-generated insights remain relevant to specific research requirements.

Furthermore, the project does not claim to provide a comprehensive solution to all issues in document analysis. Rather, its objective is to enhance human-AI interaction by redefining the manner in which LLMs are employed within academic and professional workflows. The system has been developed to enable users to solve problems by equipping them with AI-driven tools.

Based on the initial project concept, these research questions were developed:

1. **Research Question:** *"Does the TextVision interface improve the efficiency, usability, and overall user experience of document creation compared to traditional word processing tools?"*
2. **Sub-Question:** *"In what ways do the PromptDesigner and Prompt Recommendation features reduce cognitive load during the document creation process?"*

Finally, in order to ensure clarity and transparency, this documentation aims to provide an overview of the TextVision prototype and to explain the architecture and the decision behind the product. The documentation will provide a scientific foundation and an overview of the aspects in which these are implemented within the system. In order to evaluate the product, a user study was performed, additionally the results of a user study are included.

2. Fundamentals

This chapter provides an overview of the fundamental principles underlying artificial intelligence, large language models, and related fields. To fully grasp the design and implementation choices made in TextVision, it is essential to explore both scientific and technical fundamentals. The scientific section covers foundational AI concepts, including ML, DL, and human-computer interaction (HCI), while the technical section delves into distributed applications, networking protocols, and data exchange formats. Together, these elements form the basic understanding required for TextVision’s capabilities.

2.1. Related Work

The growing adoption LLMs has transformed various applications, but their full potential remains largely unrealized due to limitations in user interfaces and interaction paradigms. A significant challenge lies in prompt engineering, the process of creating effective inputs for these models, which often requires significant domain expertise and iterative refinement (Zamfirescu-Pereira et al., 2023). Current applications rely primarily on text-based interactions and fail to address the multimodal nature of real-world information processing tasks, particularly when working with complex document formats such as PDFs.

2.1.1. AI-Assisted Document Processing

Recent research has explored various approaches to improving document processing with the help of AI. Several systems have been developed to streamline research workflows (G. Lin et al., 2024; Mathur et al., 2024) and facilitate scientific work (Fok et al., 2023; S. Wu et al., 2023). For example, Scim provides an interactive system that improves the skimming of scientific papers by highlighting key sentences across different facets of research papers, enabling faster information retrieval through a customizable PDF interface (Fok et al., 2023). Similarly, ScatterShot simplifies the creation of high-quality few-shot prompts for text transformation tasks, although it lacks comprehensive prompt control mechanisms (S. Wu et al., 2023).

Beyond document processing, researchers have explored the use of LLMs as evaluators, demonstrating their potential for unbiased and scalable evaluation in multiple contexts (Desmond et al., 2024). Kim et al. emphasize the use of user interfaces as writing tools that facilitate reflection on the user’s writing process, thereby increasing engagement and productivity (J. Kim et al., 2024). Other innovations include feedback mechanisms for principle generation (Petridis et al., 2024), structured support systems for improving peer review (Neshaei et al., 2024), and multimodal systems for co-creating alternative scientific texts (Singh et al., 2024).

2.1.2. Prompt Engineering and Optimization

A critical aspect of effective LLM use is prompt engineering, an area where significant research efforts have been focused. Lin et al. present a Prompt Optimization with Human Feedback (POHF) framework that aims to find optimal prompts for black-box LLMs using

only preference feedback from users (X. Lin et al., 2024). Their approach draws inspiration from dueling bandits to select pairs of prompts for user evaluation, allowing for efficient optimization with minimal feedback instances.

Building on this foundation, Agarwal et al. present PromptWizard, a fully automated framework for discrete prompt optimization that utilizes a self-evolving, self-adapting mechanism (Agarwal et al., 2024). Through a feedback-driven critique and synthesis process, PromptWizard achieves an effective balance between exploration and exploitation, iteratively refining both prompt instructions and in-context examples to generate human-readable, task-specific prompts.

Despite these advances, Zamfirescu-Pereira et al. highlight that non-AI experts face significant challenges when attempting to design effective prompts (Zamfirescu-Pereira et al., 2023). Their study shows that users often edit prompts ad hoc in response to single observed errors, rarely systematically explore multiple conversations, and often apply human-to-human communication norms that may be suboptimal for LLM interactions. These findings underscore the need for intuitive tools that guide users through the prompt editing process.

2.1.3. Intelligent User Interfaces for LLMs

The integration of LLMs with intelligent user interfaces represents a promising direction for improving human-AI interaction (Oviatt et al., 2017). Several frameworks have emerged to facilitate this integration, with different approaches to rapid creation and refinement. Sučík et al. introduce Prompterator, a system that enables users to generate improved prompts without requiring programming expertise (Sučík et al., 2023). Similarly, Direct-GPT advances human-AI interaction by applying direct manipulation principles to LLM interactions (Masson et al., 2024). These approaches share a common goal of making LLM capabilities more accessible to non-expert users, though they differ in their specific mechanisms and target applications.

A significant challenge identified in recent research is user dissatisfaction with LLM outputs, especially among individuals with limited knowledge of these models (Y. Kim et al., 2024). This underscores the importance of designing systems that proactively support users in overcoming dissatisfaction, especially those with limited technical expertise.

2.1.4. Positioning of TextVision

TextVision integrates large language models into a multimodal interface for document analysis and authoring, building on current research in prompt engineering and LLM interfaces. Current tools often use generic prompts that do not address specific use case requirements. TextVision addresses this limitation by allowing users to design, refine, and save custom prompt shortcuts, thereby increasing the efficiency of user-LLM interaction (Zamfirescu-Pereira et al., 2023).

The system uses an AI-based knowledge management component with context-aware capabilities. This component is built on a composite architecture that incorporates both

proprietary and open source LLM variants, including models specifically tuned for scientific research applications (Hu et al., 2021). This implementation is consistent with research showing the importance of domain-specific adaptation for optimizing LLM performance.

TextVision incorporates methodologies from recent developments in structured approaches to prompt creation, including frameworks such as PromptWizard (Agarwal et al., 2024) and techniques for human feedback-based prompt refinement (X. Lin et al., 2024). By combining the principles of Prompterator (Sučik et al., 2023) and DirectGPT (Masson et al., 2024), TextVision provides advanced prompting features in addition to features for scientific work. These include dynamic prompt recommendations based on user input and a Prompt Designer for systematic prompt creation and optimization.

TextVision addresses the limitations of existing AI-based document processing tools through its context-aware, multimodal interface that improves the accessibility and efficiency of LLM capabilities for document processing. The system facilitates iterative, real-time refinement and integration of AI-generated output into user workflows, advancing human-AI collaboration in academic and professional document processing contexts.

2.2. Foundations of Artificial Intelligence

In order to understand how LLMs and generative AI work, the basics of artificial intelligence are first introduced. The definition and origin of the term "artificial intelligence" are presented, followed by a classification of strong and weak AI and a description of its characteristics. This serves to create the theoretical basis for the introduction to the sub-areas of AI, including machine learning, deep learning, and the associated neural networks (NN) (see Figure 1).

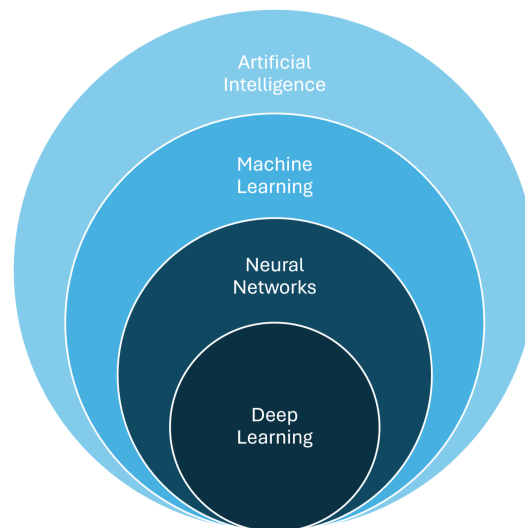


Figure 1: Basic concepts of AI according to Sonnet, 2022

Although AI has only recently become a popular social topic, it has existed in smaller forms for decades (Harwardt & Köhler, 2023). In order to define the term, its origins must be considered. One significant early event was the "Summer Research Project on Artificial Intelligence", which took place at Dartmouth College in Hanover (New Hampshire) in 1956 (Harwardt & Köhler, 2023). The Dartmouth conference has since been regarded as the birth of AI. The key figure behind this event was John McCarthy, who was convinced that machines could be developed that were capable of reproducing every aspect of human intelligence. Based on this key event, the concept of AI has become increasingly widespread, although a uniform definition has never been established. The reason for this lies in the unavoidable link between the term "intelligence" and its definition. The term artificial implies that something is to be mapped and that the variable to be mapped is intelligence. However, there are different understandings of how the term intelligence and the associated abilities should be defined, which is why it is difficult to come up with a uniform definition. A current and comprehensive definition has been drawn up by the German Association for Information Technology, Telecommunications and New Media (Bitkom) in cooperation with the German Research Center for AI (DFKI). It describes AI as IT applications that aim to exhibit intelligent behavior, which can be defined on the basis of various capabilities. The skills required for this are perception, understanding, action and learning (Harwardt and Köhler, 2023).

In the context of AI, the terms strong and weak AI (*Weak and Strong AI*) are frequently encountered. The term "strong" suggests a superiority over weak AI. Strong AI systems are designed to think and act at least at a human level (BITKOM, 2017).

The aim is to map and imitate processes in a person's brain. This also includes aspects such as consciousness or empathy (Buxmann and Schmidt, 2021).

If such a system were to emerge and continue to develop and improve, it would reach the state of a so-called singularity. This refers to the state in which AI surpasses human intelligence. However, this state has not yet been fully achieved, which is why today's AI systems tend to be associated with the term weak AI (BITKOM, 2017).

Weak AI, also known as *Narrow AI*, does not attempt to imitate the entire human way of thinking or behavior, but develops algorithms for specific and isolated problems. The ability to learn is a fundamental requirement for both weak and strong AI. This requirement is fulfilled by ML and, more specifically, by DL (Buxmann and Schmidt, 2021).

2.2.1. Machine Learning

Today's computers and programming are based on algorithms. In simple terms, an algorithm is a sequence of specific instructions that are to be carried out with the aim of generating an output from a specific input. Machine learning is a branch of AI and deals with the development of algorithms. These algorithms should have the ability to learn from data and make decisions independently. Machines should be able to learn continuously through experience and constantly improve their performance in certain tasks (Bhattacharyya et al., 2020).

The ability to learn from information and thus constantly evolve is a characteristic that is also attributed to humans. Machine learning is the basis for creating a "strong AI", which was already mentioned in the previous chapter. Only when an intelligence can develop independently can it be described as strong. The aim is to be able to perform tasks that were previously only possible for humans. This can include the identification of people and objects, a task that humans can easily master (Bhattacharyya et al., 2020).

The algorithms developed as part of ML analyze large amounts of data in order to identify patterns and correlations. Predictions or decisions are made on the basis of this collected information. The strength of these algorithms lies in their ability to constantly learn and improve by learning from the results of the predictions and continuously adapting the models accordingly. A basic distinction is made between three types of machine learning, which are discussed below (Bhattacharyya et al., 2020).

The first type of machine learning is supervised learning. Here, models are trained with labeled data where the correct answer is already known (see Figure 2). The model learns to map input data X to output data Y in order to be able to make predictions for new, unknown data (Bhattacharyya et al., 2020).



Figure 2: Supervised Learning according to Buxmann and Schmidt, 2021

In contrast, there is the concept of unsupervised learning (see Figure 3). In this method, models are trained with unlabeled data. The aim is to discover structures or patterns in the data without a correct output being specified. Consequently, only the input data is available and, compared to supervised learning, it is an explorative approach that is therefore less controllable. The aim is for the algorithm itself to recognize regularities and structures (Bhattacharyya et al., 2020).



Figure 3: Unsupervised Learning according to Buxmann and Schmidt, 2021

The third type of learning is reinforcement learning (see Figure 4). Here, an agent learns how it should behave in an environment in order to achieve a specific goal. This agent receives feedback in the form of rewards or punishments and develops a strategy through trial and error to maximize the reward. The agent must learn which actions deliver the best results in which situation. It is therefore not surprising that this type of learning is used in the context of artificial intelligence for board games. It is more about the sequence of different actions that lead to a tactic than about individual actions (Bhattacharyya et al., 2020).

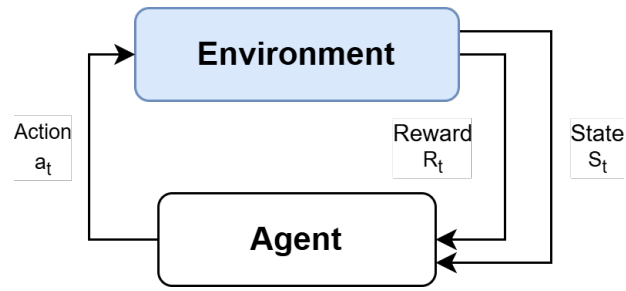


Figure 4: Reinforcement Learning according to Buxmann and Schmidt, 2021

2.2.2. Neural Networks

The basic idea behind neural networks is to create systems that can mimic the structure and functioning of the human brain. Such networks consist of numerous interconnected processing units. These units are called neurons and are arranged in layers. Each layer receives information from the previous layer, processes it and passes it on again to the next layer. This creates complex nested networks that are able to recognize complex patterns in data and learn from them. The learning ability of these networks is based on the adaptation of the connection strengths between the neurons during training (Buxmann and Schmidt, 2021).

The first neural network, the "Rosenblatt perceptron", which was named after its creator Frank Rosenblatt, is the forerunner of complex neural networks (Rosenblatt, 1958). It consists of a neuron that receives input values and can generate an output value from them (see Figure 5) (Sonnet, 2022).

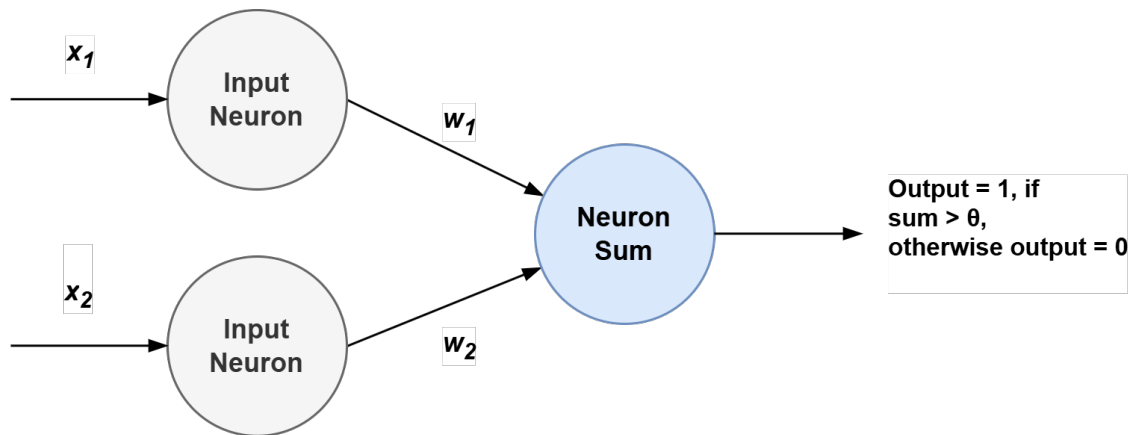


Figure 5: Simple perceptron according to Sonnet, 2022

In the case of such a simplified perceptron, there is an input x , which contains the input values. Each input is assigned a weight w and multiplied by this weight. These weights are the parameters of the perceptron, which represent the strength of the connection between

neurons. Depending on the number of inputs, the weighted inputs are summed to produce an output (Sonnet, 2022). A formula corresponding to this procedure is:

$$y = \sum_{j=1}^d w_j x_j + w_0.$$

It then checks whether the weighted sum is greater than a previously defined threshold value θ . If this is the case, the neuron outputs a 1, in any other case a 0 (Sonnet, 2022).

Output = 1 if $x_1 w_1 + x_2 w_2 > \theta$, otherwise output = 0.

2.2.3. Deep Learning

The enhancement of the simple perceptron is the multilayer perceptron (MLP). As the name implies, unlike the Rosenblatt perceptron, this involves several layers of neurons. The already known input layer is followed by one to any number of additional hidden layers, called *Hidden Layer*. The more common term for MLP is artificial neural network, also known as ANN. The large number of layers in an ANN makes it possible to solve more complex problems, as non-linear patterns in data can be recognized. This resolves the limitations of simple perceptrons, which can only approximate linear functions (Alpaydin, 2022).

The current enthusiasm for AI is largely driven by ANNs. The depth of a NN, determined by the number of layers, enables DL. While there is no strict definition of "deep", it generally refers to a network with multiple layers. However, the exact threshold remains undefined (Kirste and Schürholz, 2019).

Such NN (see Figure 6) is called a *feedforward* network due to the direction of the information flow. However, there are also networks in which the information flow does not only go in one direction, but is fed back or the information is fed back to the previous layer. If such feedback is used, it is a recurrent network (Kirste and Schürholz, 2019).

An ANN is characterized by the transfer of values from previous layers. As with the simple perceptron, these values include the associated weights and threshold values of the neurons. The arbitrarily strong nesting creates a branched network that can also process complex information (Kirste and Schürholz, 2019).

Each neuron receives input from the previous neurons, then determines its activation and outputs a corresponding value when activated (Sonnet, 2022).

However, in order to use such a network effectively, it must also be trained, as it initially has no weights. One possible method for this is *Backpropagation*, which belongs to the already mentioned learning type of supervised learning. The training of such a network is significantly more complex and resource-intensive than the actual use (Kirste and Schürholz, 2019).

Backpropagation is based on the chain rule of differential calculus and makes it possible to propagate an error from the output backwards through the ANN and adjust the weights based on this. This procedure from back to front also gave rise to the name for the procedure: Backpropagation. The aim is to calculate the gradients of the error function in the network. These gradients are used to specify the extent of the weight adjustment in

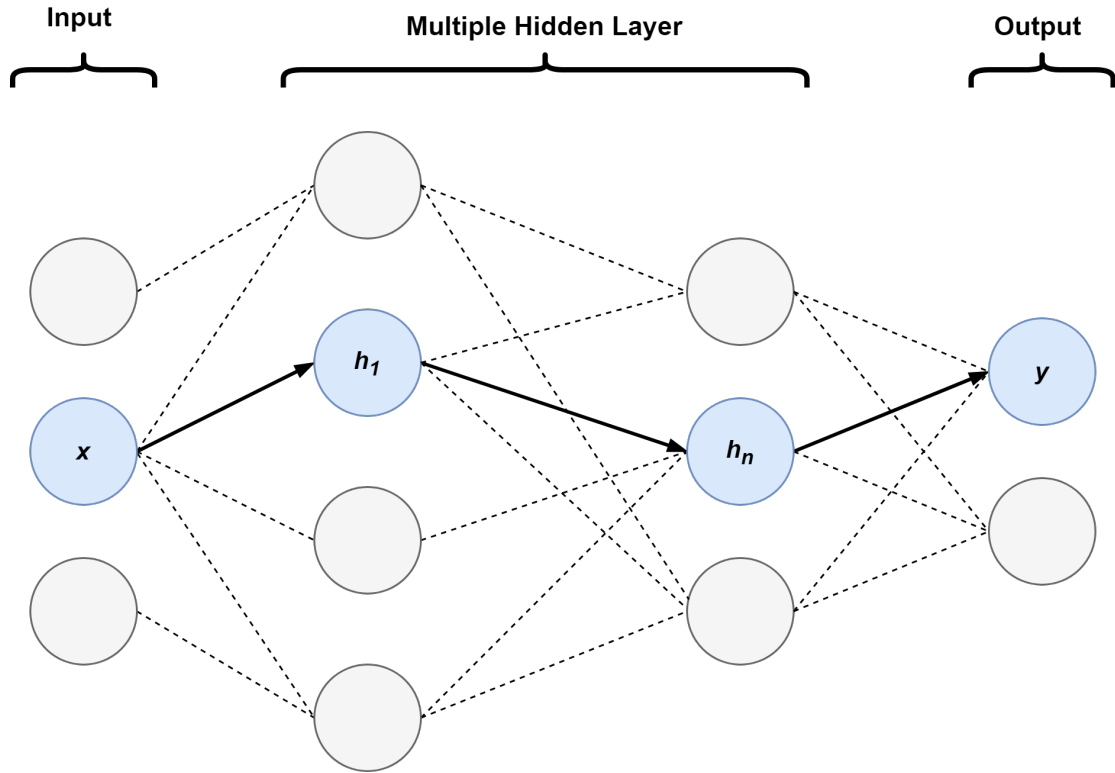


Figure 6: Artificial Neural Network according to Buxmann and Schmidt, 2021

order to minimize the errors of a prediction. This iterative process is intended to ensure continuous improvement in the accuracy of predictions (Sonnet, 2022).

2.2.4. Generative AI & Natural Language Processing

The basics of AI and possible learning methods such as backpropagation have already been discussed in the introduction. For LLMs, however, methods of generative AI and Natural Language Processing (NLP) play a particularly important role.

Conventional AI typically refers to discriminative AI. Discriminative AI is designed to make distinctions between different data sets. They focus on classifying data sets and learning the probability with which features contribute to class assignment. Predictions about new data are made on this basis (Dahm and Zehnder, 2023).

On the other hand, there is generative AI, which is also known as a generative model and is also a specialized form of AI. One example of such generative models are so-called Generative Adversarial Networks (GANs). GANs are designed to generate new data that is similar to the training data. They consist not only of one neural network, but of two competing networks. A generator, which produces new data, and a discriminator, which attempts to distinguish between generated and real data. The aim is for the generator to

produce such realistic data that the discriminator can no longer differentiate whether the data is real or not (Bhattacharyya et al., 2020).

Generative AI is therefore important for LLMs as it enables the creation of new, data-driven and contextually relevant texts.

NLP serves as an interface between AI and human language. In general, NLP can process spoken or written natural language. Processing not only means that the string of characters is used as input, but also that the meaning of a text can be understood and new output generated based on the meaning and context. The generated text should be comprehensible, authentic in style and grammatically correct. The ability to process natural language means that AI is now also accessible to people who do not know programming languages (Harwardt and Köhler, 2023).

NLP is subdivided into Natural Language Understanding (NLU) and Natural Language Generation (NLG). NLU is concerned with understanding human language, i.e. grasping the meaning of words and sentences in context and understanding the intentions behind statements. NLG, on the other hand, is the process of generating natural language from data. It enables computers to express information and thoughts in a way that is understandable and natural to humans. This allows AI to interact with users (Harwardt and Köhler, 2023).

Due to the extended interaction and application possibilities, these technologies are used in large language models, e.g. within a chatbot such as ChatGPT from OpenAI (Plank, 2023) or Gemini from Google (Google, 2024).

The NLP process is a multi-stage process that is divided into four main phases. These include morphological, syntactic, semantic and contextual analysis (see Figure 7).

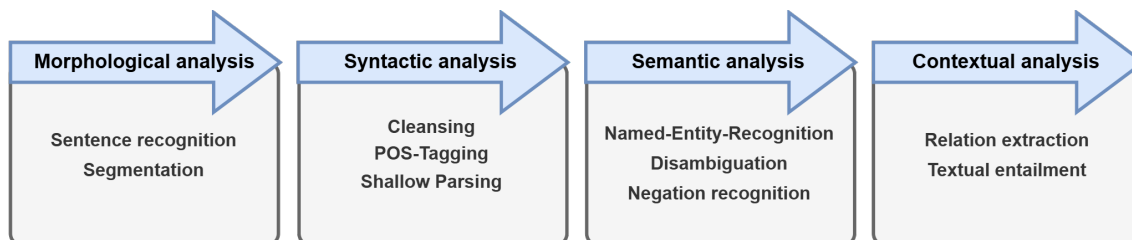


Figure 7: NLP process according to Jungmann et al., 2018

Each of these phases involves specific tasks that contribute to the processing and understanding of natural language by computers. Morphological analysis uses sentence recognition, to identify the boundaries of sentences within a text, and segmentation, to break text into smaller units such as words (Jungmann et al., 2018).

The syntactic analysis optimizes after the previous phase through error correction, Part-of-Speech-Tagging and Shallow Parsing. It removes disruptive elements, classifies words according to word types and identifies basic sentence structures to make texts more comprehensible and accessible for further analysis (Jungmann et al., 2018).

Subsequently, semantic analysis deepens the understanding of texts in natural language processing by identifying specific entities, clarifying ambiguities and interpreting nega-

tions. Important information such as proper names is extracted through Named-Entity-Recognition. Disambiguation ensures clear meanings of words in context, and negation recognition is crucial for the correct interpretation of the overall statement (Jungmann et al., 2018).

The final phase is contextual analysis. Here, relationships between entities and concepts are identified and interpreted to enable a deeper understanding of the text. Through relation recognition and textual inference, both explicit and implicit information and connections within the text are uncovered (Jungmann et al., 2018).

2.2.5. Large Language Models and Multimodal Models

Large Language Models are large generative language models that use AI-Technologies. They can be used to generate text and are a form of generative AI. As the name suggests, these models are characterized by the large amount of data they have been trained with and have to deal with (Luber, 2023).

LLMs use the technologies already discussed, such as ANNs, DL and NLP. These are used to process, understand and generate natural language. Although the LLMs are trained with a large amount of data, they are fine-tuned for specific tasks. The amount of data sets them apart from simple generative language models, as they can understand more complex texts, questions and instructions. This is made possible, among other things, by the number of parameters on which they are based. For most LLMs, this number of parameters amounts to several hundred billion.

While the main competence of LLMs lies in processing text, there are also some multimodal models, which means that they can also process content such as images, videos or audio data. In essence, LLMs function by making predictions and calculating probabilities. They determine the most probable continuation of the text for each subsequent word during the process of text generation (Luber, 2023).

In addition to the already known KNNs and deep learning, such models are often based on the Transformer architecture, which is examined in more detail in the following passage. The Transformer architecture (see Figure 8) represents the current state-of-the-art foundation for modern LLMs. It marks a significant advancement in natural language processing. This architecture originates from the publication of the paper "Attention Is All You Need" by Ashish Vaswani et al. in 2017 (Vaswani et al., 2017).

The core innovation of this architecture is the extensive use of attention mechanisms. These allow for simultaneous processing of dependencies between all words in a sequence. This capability distinguishes the Transformer architecture from known approaches like recurrent neural networks (RNNs). Previous networks performed sequential calculations and thus struggled to efficiently process long-range dependencies (Amaratunga, 2023).

The architecture can be divided into three main components: the tokenizer, the embedding layers, and the Transformer layers (see Figure 8). Tokenizers are responsible for converting input text into tokens. In the embedding layers, tokens are given semantic meaning through representations. The Transformer layers perform the actual processing and reasoning over the data. They consist of attention mechanisms and multi-layer perceptrons.

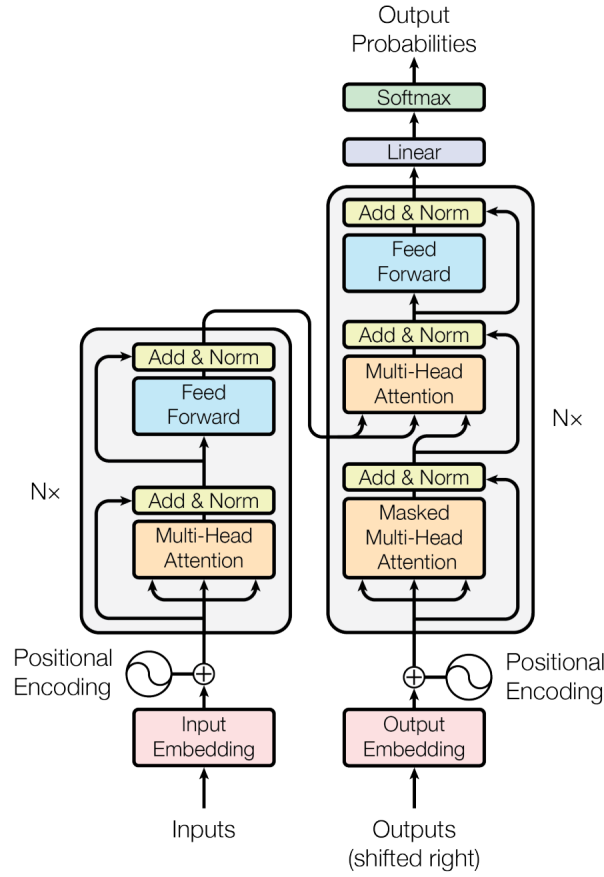


Figure 8: Transformer Architecture (Vaswani et al., 2017)

The uniqueness of the Transformer architecture lies in these Transformer layers. Two types of layers are distinguished: the encoder and the decoder (Amaratunga, 2023). The encoder's task is to process the input sequence and generate a series of representations. The goal is to capture the information and relationships between various parts of the input sequence. These representations contain the meaning of individual words and the context in which they are used. The tasks are divided within the encoder, which consists of a stack of N identical layers. Each of these layers has two sublayers: a self-attention mechanism and a feed-forward network. The self-attention allows the encoder to understand the relationships between all words in the input sequence by calculating how each word relates to every other word in the sequence, thus capturing the context of each word. The feed-forward network follows the self-attention layer and applies non-linear transformations to the data, generating a final representation of the input sequence. Additionally, positional encoding is employed as a third step to consider the order of words in the input sequence, which is necessary because the Transformer architecture itself has no information about sequence order (Amaratunga, 2023).

In contrast, the decoder works similarly to the encoder and is also built from N identical stacked layers. It is designed to use the representations generated by the encoder to produce an output sequence. In addition to the two sublayers known from the encoder, the decoder has a third layer that functions as an encoder-decoder attention layer. Through this additional layer, the decoder can utilize the outputs of the encoder to generate an output sequence (Amaratunga, 2023).

The attention mechanism in Transformer models allows them to focus on specific parts of the input data while generating output. This enables the model to pay attention to certain information from the input data.

This works by assigning different weights to various parts of the input sequence. There are different types of Attention, among which Scaled Dot-Product Attention and Multi-Head Attention are particularly noteworthy (Amaratunga, 2023).

Scaled Dot-Product Attention was developed to solve the vanishing gradient problem. This problem occurs when the gradient in backpropagation becomes increasingly smaller, to the point where the network can no longer learn. It arises when traditional attention mechanisms work with long sequences. The result of the increasingly small gradient is that weight updates become negligibly small, making the learning process pointless beyond a certain point (Amaratunga, 2023).

Scaled Dot-Product Attention works with three main components: Queries (Q), Keys (K), and Values (V), which are derived from the input data. By calculating the dot product of the Query with all Keys, an attention score is computed. To avoid potential scaling issues, the result is divided by the square root of the dimension of the Keys. Then, a softmax function is applied to convert the scores into probabilities. These probabilities determine how much each value contributes to the output. The scaling prevents the softmax function from operating in areas with too small gradients, which addresses the original vanishing gradient problem (Amaratunga, 2023).

The Scaled Dot-Product Attention function according to Vaswani et al., 2017 is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q is the matrix of Queries, K the matrix of Keys, and V the matrix of Values. The product QK^T is the dot product of the Query matrix Q with the transposed Key matrix K^T . In the denominator, d_k is the dimension of the Keys. The division by $\sqrt{d_k}$ serves for scaling to prevent the softmax function from operating in areas with very small gradients, which can improve the stability of the learning process (Vaswani et al., 2017).

In contrast, Multi-Head Attention is an extension of Scaled Dot-Product Attention. As the name suggests, Multi-Head Attention divides the attention mechanisms into multiple heads, which are executed in parallel. Each of these heads can direct attention to different information, leading to parallelism. This allows different information to be processed in parallel, enabling a more multi-faceted analysis of the input data. Subsequently, the results of the various heads are combined.

The Multi-Head Attention function according to Vaswani et al., 2017 is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where each head head_i is defined as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

In this formula, W_i^Q , W_i^K , and W_i^V are weight matrices for the linear transformations of the Queries, Keys, and Values for the i -th head, and W^O is another weight matrix used to combine the concatenated outputs of the individual heads.

Tokens are the smallest units of a text that can be processed using NLP. They are created in the tokenizing process and are usually words, parts of a word or punctuation marks. Tokenizing refers to the process that is necessary to create these tokens. This usually refers to the decomposition and conversion of raw text into a form that can be understood by NLP models. Without tokenizing, models such as transformer models would find it difficult to understand the structure and meaning of a text, as there would be no clear separation between the individual elements. Depending on the intended use and type of token, there are different techniques for carrying out the tokenizing process. The most common techniques are whitespace tokenizing, punctuation tokenizing, word tokenizing and subword tokenizing. These techniques are described by way of introduction in Figure 9 (Amaratunga, 2023).

Whitespace	Punctuation	Word	Subword
The text is divided into tokens using spaces. Simple and frequently used, but can cause problems in special cases such as compound words or abbreviations.	The text is divided into tokens using punctuation marks. Useful for texts with many punctuation marks, but can lead to problems with abbreviations or other special cases.	Uses language-specific rules to divide the text into words. Can deal with special cases such as compound words, contractions and punctuation in more detail.	Methods such as Byte-Pair-Encoding (BPE) and SentencePiece divide words into subword units. Allows the model to deal with words that are not in the vocabulary and to handle rare or unknown words more effectively.

Figure 9: Tokenizing techniques according to Amaratunga, 2023

As already discussed in previous sections, LLMs can process content in the form of texts and generate new texts based on this. However, this property very quickly limits the type of input, so that a corresponding further development of such models followed. The further development of LLMs are Large Multimodal Models (LMMs). This further development combines generative AI with multimodal AI. In addition to text, multimodal models can also record other forms of information such as image or audio content (Erik, 2024).

LMMs are also referred to as Multimodal Large Language Models (MLLMs). Such models have been developed with the aim of overcoming limitations of pure text-based LLMs such as GPT-3, BERT and RoBERTa. These models have difficulties in understanding and processing other data types. With the ability to process multimodal data, MLLMs can be used in high-value domains such as multimodal robotics, document intelligence and robotics technology. An example of such a model is GPT-4 from OpenAI, which can process both images and text and achieves human-like performance in various benchmark tests. This multimodal perception is crucial for the enrichment of knowledge and interaction with the real world (J. Wu et al., 2023).

Despite this further development, the term LLM is still widely used, so that multimodal models are also simply referred to as LLM. As a result, articles and literature do not necessarily differentiate between LLM, LMM or MLLM.

2.2.6. Prompt Engineering

Prompt engineering refers to the systematic process of designing, optimizing and specifying input instructions (prompts) for generative AI models in order to generate specific and relevant outputs (Amaratunga, 2023; Hein et al., 2024). Methods such as the integration of clear instructions, contextual data, or examples are used to guide model behavior without time-consuming fine-tuning of the underlying AI architecture (Bsharat et al., 2023). A prompt typically comprises a combination of task, questions, context data and optional examples (Amatriain, 2024). It ranges from simple questions ("Name three facts about X") to structured techniques such as Chain-of-Thought (CoT), where the model is guided through step-by-step thinking instructions to more complex solutions (Amatriain, 2024; Fagbohun et al., 2024). This strategic prompt design phase makes it possible to adapt models to domain-specific tasks through clever formulations, thereby increasing output quality and consistency (Sahoo et al., 2024).

One of the key challenges is the balance between overly general and overly restrictive instructions. While overly detailed specifications can limit creativity, imprecise instructions lead to irrelevant results (Sahoo et al., 2024). Efficient prompt engineering has been demonstrated to unlock the full potential of LLMs, particularly in applications where task adaptation without model retraining is critical (Fagbohun et al., 2024; Knoth et al., 2024).

2.2.7. Fine-Tuning

In the previous chapters, the general concept and training process of LLMs or, to a broader extent, transformer-based architectures, was described. LLMs are trained on massive text datasets and oftentimes consist of several billion parameters (Minaee et al., 2024, p. 1) and commonly require thousands of training hours (Sani et al., 2024, p. 2), making it difficult to efficiently train them from scratch. Thus, if there is a given task that is not currently supported by already available LLMs or not supported to a desired extent, training an LLM towards that goal from scratch is usually not a viable option.

Therefore, in many cases, it can be a feasible approach to further train an already pre-trained model towards a specific learning goal, to help increase the model's performance with respect to that goal. This specific form of training an already pre-trained model is referred to as fine-tuning and is considered an effective way of improving model performance as well as mitigating or even removing undesirable behaviors in that model (Dettmers et al., 2023, p. 1). Fine-tuning is a subfield of transfer learning. Transfer learning, in the context of machine learning, describes the goal of making the learned knowledge of one specific task applicable to use for another task, thereby reducing training time, since certain parts of the model are already trained. The tasks, however, need to be somewhat similar in nature (Vrbančič and Podgorelec, 2020, p. 1). Thus, while a model is initially trained on new data, the trained model is then trained again on different data that relates to a more specific task or problem. To avoid re-training the entire model from scratch, usually only a subset of layers is trained during fine-tuning, while the rest are excluded from that process. Those excluded layers are usually referred to as frozen layers and are used to retain and leverage a significant proportion of the model's already acquired knowledge. Still, there are also some downsides to fine-tuning: It can potentially be difficult to determine the layers that should be frozen during the training process and those that should be trained. Also, during fine-tuning, there is no fixed set of optimal hyper-parameters that is available from the beginning (Vrbančič and Podgorelec, 2020, p. 1). Instead, parameters usually need to be approximated during the fine-tuning process. Another problem is that, even though fine-tuning might not be as cost intensive as completely training a model from scratch, it oftentimes still requires huge amounts of available resources, considering the huge numbers of trainable parameters in most models (Dettmers et al., 2023, p. 1).

The most basic way to fine-tune a pre-trained model is called full fine-tuning. With full fine-tuning, the entirety of the model's parameters is updated during the fine-tuning process. Due to the huge size of many models, this is often an inefficient and unsustainable approach (Dettmers et al., 2023, p. 3). To tackle this problem, a technique called Parameter-Efficient-Fine-Tuning (PEFT), was developed. PEFT is composed of several different approaches that aim to reduce the parameter count that needs to be updated during finetuning while leaving the rest unchanged (Olan, 2003, p. 1). Some prominent PEFT approaches include:

- **Low-Rank Adaption (LoRA):** Instead of updating all the model parameters, this approach adds low-rank matrices to the existing parameters. During the fine-tuning process, only the low-rank adaptation matrices are updated, while the original

model's parameters remain frozen, thereby significantly reducing the trainable parameter count and storage, as only the matrices need to be stored (Dettmers et al., 2023, p. 3). LoRA is said to lower the hardware barrier to entry by up to 3 times (Hu et al., 2021, p. 2). The downside of LoRA is a slight decrease in expressiveness compared to full finetuning, since the actual model parameters are not updated.

- **Quantized Low-Rank Adaption (QLoRA):** QLoRA is a different form of LoRA, which aims to reduce the required memory even further. This is achieved by loading the base model in 4-bit precision (NF4), while model parameters are usually stored in 16-bit floating-point (FP16) or bfloat16 (BF16). This drastically reduces the VRAM usage during the finetuning process and thus even allows larger models to be trained on consumer GPUs (Dettmers et al., 2023, p. 4). Though this might come at the cost of lower model precision, due to information loss as a result of the 4-bit quantization.
- **Prefix Tuning:** Prefix-tuning is an approach that refrains from even modifying the model. Instead, trainable word embeddings that are generally not in the model's vocabulary are prepended to the input tokens (Hu et al., 2021, p. 6). These prefixes then in turn influence the model's attention mechanism, without actually altering the model parameters (Han et al., 2024, p. 6). Due to its nature, this can potentially be highly efficient, as no model parameters are updated.
- **Side-Tuning:** With this approach, a small side network is trained next to the frozen model. Its outputs are then later merged with the output of the original model. Thus, instead of updating the initial model, the side network learns the new task-related features (Han et al., 2024, p. 12). This approach therefore also comes with a certain scalability in the context of new tasks, since it simply requires new side networks to be trained. A possible downside might be computational cost, as a second, albeit smaller, model is part of the training process.

2.2.8. Introduction to Chatbots

Chatbots are AI-driven software applications designed to simulate human-like conversations with users through text or voice interactions (Figueroa-Torres, 2025). Their core functions are to understand user input, generate natural language responses, and assist users in completing a variety of tasks, from simple customer support to complex problem solving (Casheekar et al., 2024). The main purpose of chatbots is to enhance user experience through instant and effective help and reduce human intervention in daily tasks (Casheekar et al., 2024).

Main Functions and Working Principles

The working principle of chatbots is based on NLP and deep learning technology, and their architecture usually includes the following three main modules:

- **Natural Language Understanding:** responsible for parsing user input, identifying intents and entities. For example, when a user asks "What's the weather in Bangalore?", the NLU module extracts the intent of "query the weather" and the entity of "Bangalore".
- **Dialogue Manager:** Records the dialogue state, including the user's question and the chatbot's answer, and decides how to respond to the user's next query based on this information.
- **Natural Language Generation (NLG):** Generates natural language responses using the state information of the knowledge base and the dialogue manager to ensure that the answers are relevant to the user's query and easy to understand (Casheekar et al., 2024).

In recent years, chatbots have gradually evolved from rule-based systems to generative AI, which can generate more natural and intelligent conversations. For example, the LoRA-LiteE framework combines supervised fine-tuning (SFT) and low-rank adaptation technologies to achieve efficient preference adjustment under limited resources, improving the scalability and accessibility of chatbots (Yang et al., 2024).

In addition, emotional factors also play an important role in user experience. Modern chatbots improve user satisfaction and engagement through enhanced emotional interaction capabilities (Casheekar et al., 2024).

2.2.9. Human Computer Interaction

Another crucial aspect that significantly contributes to the success of software is Human-Computer Interaction (HCI). It deals with the design, analysis, and implementation of interactive computer systems for human use and the related phenomena (Hewett et al., 1992). The primary goal of HCI is to optimize usability, efficiency, and user satisfaction by combining insights from disciplines such as computer science, psychology, design, and ergonomics.

There are six ways in which a human can interact with an AI (see Figure 10). These possibilities are:

Humans interact with AI systems in a variety of ways. One approach is through voice input, as exemplified by voice assistants such as Siri that respond to spoken commands. Graphical user interfaces further facilitate communication by visually presenting information in an accessible way. In addition, gesture control enables intuitive interaction, with movements captured and analyzed by cameras. Body language and facial expressions provide additional contextual information. Physical actions, such as turning a steering wheel to steer a vehicle, also serve as input mechanisms. In addition, AI systems can use biometrics, such as eye tracking, to respond dynamically to human reactions.(Asemi et al., 2022).

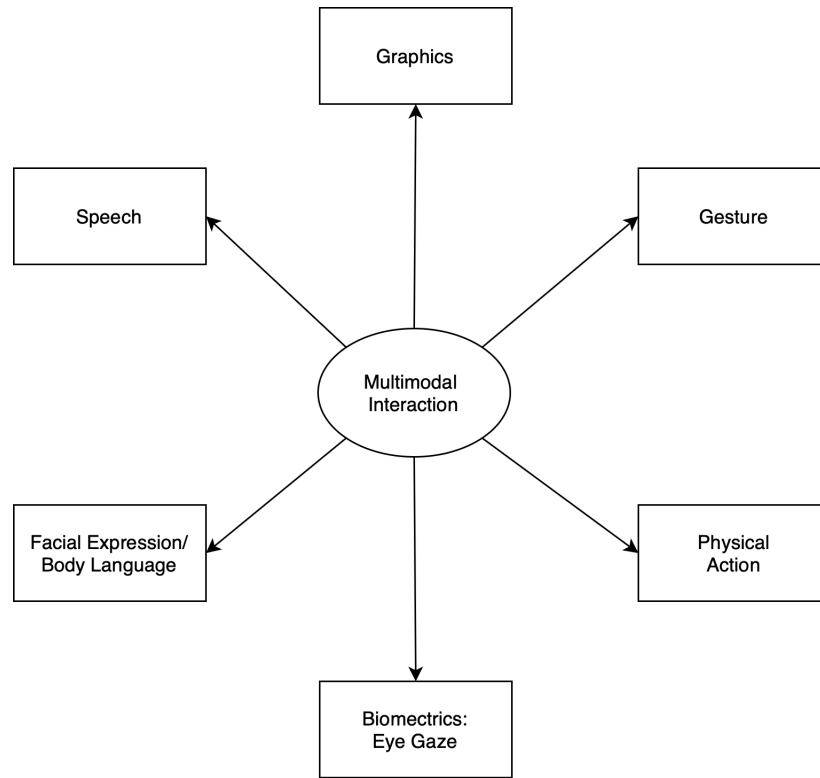


Figure 10: Ways of Interaction (Oviatt et al., 2017)

Human-computer interaction is influenced by many factors. To ensure the success of a system, it is therefore essential to optimize these aspects in a targeted manner. By applying established principles and best practices, different areas of a system can be improved. A key approach in HCI is Human-Centred Design (HCD), which places the user at the centre of the application (Asemi et al., 2022). From the earliest stages of development, the needs of users are prioritized. They are actively involved in the process, for example through interviews or surveys. This continuous evaluation allows feedback to be taken into account at an early stage and integrated into development. By repeating this process iteratively, the final product can be optimized step by step.

2.2.10. Intelligent User Interfaces

The concept of (IUIs) is central to the development of our multimodal interface for working with documents. IUIs aim to incorporate intelligent automated capabilities in human-computer interaction, with the goal of improving performance and usability in critical ways (see Figure 11) (Oviatt et al., 2017).

The increasing use of AI in software systems means that more and more users are coming into contact with this technology. AI enables the integration of new features into existing software or the development of entirely new systems. It opens up new ways of interacting,

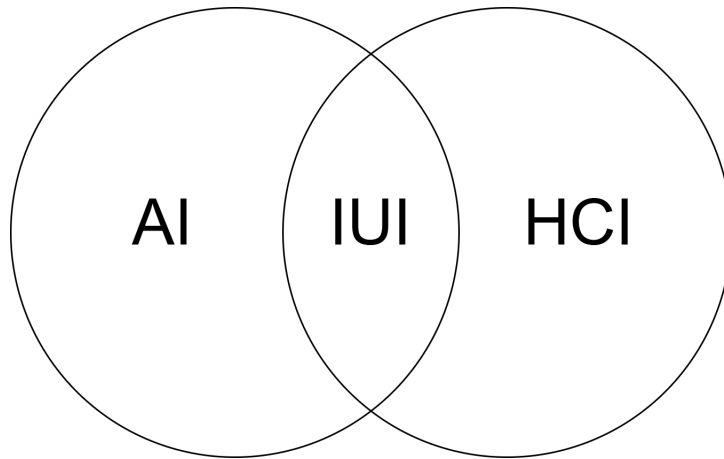


Figure 11: Connecting AI and HCI with IUI (Oviatt et al., 2017)

for example through chatbots or voice control. Systems should align with this vision, particularly in its integration of AI-powered features for document analysis and creation. In general, IUIs should effectively leverage human skills and capabilities, enabling human performance with an application to excel (Oviatt et al., 2017). This principle will be reflected in our interface design, which will allow users to interact with documents through various modalities, including text selection, voice commands, and images included in the uploaded PDFs.

Furthermore, our interface incorporates something termed "collaborative multimodality" (Oviatt et al., 2017). This concept is evident in the way our system allows users to work collaboratively with the AI-Assistant, highlighting text for analysis, asking questions and receiving context-relevant suggestions. The AI-Assistant acts as a collaborative partner in the document creation and editing process, showcasing human-AI collaboration for IUIs. Overall, our multimodal interface for LLMs embodies many of the key principles in the field of IUIs, and represents an efficient and collaborative human-computer interaction approach in the domain of document processing and creation.

2.2.11. Multimodality

Multimodality refers to the integration of multiple information input and output methods in AI systems to more comprehensively simulate human perception and interaction capabilities. These modalities include but are not limited to text, voice, image, video, etc. The core of multimodal technology is to improve system performance and user experience by integrating information from different modalities (Sheng et al., 2025).

In the field of AI, the application of multimodal technology is becoming more and more extensive. For example, UniSpeaker is a multimodal-driven speech generation method that achieves speech synthesis that is closer to the target speaker by mapping different speech description modalities to a shared speech space (Sheng et al., 2025).

Multimodality vs. Multimedia

Multimodality refers to the integration of multiple interaction modalities (such as text, speech, gestures) to create a more interactive and user-friendly interface. In TextVision, multimodality includes speech-to-text, text-to-speech, and voice commands, with the focus on interaction. (Sheng et al., 2025).

Multimedia refers to the use of multiple content types (e.g., text, images, videos) in a single medium, with the focus on content types (Anshari, 2024).

Importance of Speech Modality in Multimodality

Speech modality is a crucial component of multimodality, especially in human-computer interaction and intelligent systems. Speech interaction not only provides a more natural and intuitive user experience but also enhances system intelligence through emotion recognition and semantic understanding (M. Khan et al., 2025).

For example, MemoCMT is a multimodal emotion recognition system that combines speech signals with text transcriptions. It uses deep learning techniques like HuBERT and BERT to extract features and employs a cross-modal fusion strategy to recognize emotions. This system outperforms most single-modality emotion recognition methods on datasets like IEMOCAP and ESD, demonstrating the advantages of multimodal fusion in emotion recognition (M. Khan et al., 2025).

Moreover, multimodal technology has been widely applied in speech emotion recognition. For instance, the ISSA-BiGRU-MHA method combines Mel-frequency cepstral coefficients (MFCCs) from speech and Glove word vectors from text to achieve multimodal emotion recognition. Experimental results on multiple datasets show that multimodal fusion achieves higher accuracy and robustness in emotion recognition tasks (Shang and Fu, 2024).

2.3. Technical Fundamentals

The implementation of TextVision relies on a robust technical infrastructure. This section outlines key technical concepts that support TextVision’s architecture, including the client-server model, communication protocols like HTTP and WebSockets, and data formats such as JSON and JSON Web Tokens.

2.3.1. Distributed applications - Client-Server Model

A distributed application is based on the principle of decentralisation. Instead of a monolithic structure, independent software components are developed and run on different systems. on different systems. The Client-Server Model is a typical type of such architecture (Wagenknecht, 2004, p. 117 seqq.).

The Client-Server Model consists of a client and a server. The client sends requests to the server, which processes them and answers correspondingly (Oluwatosin, 2014). The server acts as a service provider and a distinction can be made between a web server and a backend server. The web server provides static resources such as HTML and Javascript code that can be interpreted by a client-based browser, while the backend server provides

access to dynamic resources and often contains a database for persistent storage and management of information. Both software applications run on a physical computer, allowing parallel operations and it is possible that clients and server are running on the same computer (Wagenknecht, 2004, p. 117 seqq.). Multiple clients can use the service of a single server while they have no knowledge about each other (Baun et al., 2015). The following Figure 12 illustrates a simplified form of the Client-Server Model.

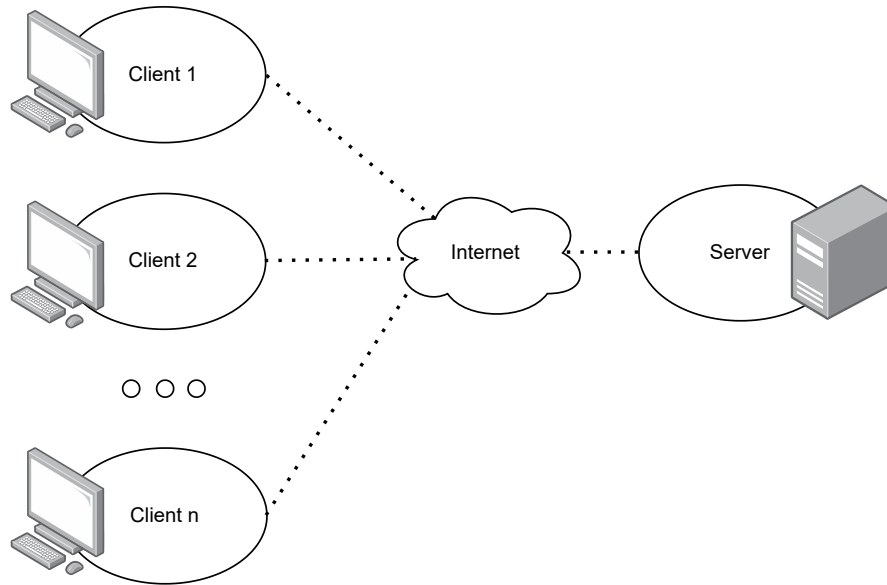


Figure 12: Client-Server Model

The inter-process communication between client and server involves an exchange of data (Oluwatosin, 2014). This exchange happens via an interface defined by a protocol. The following sections will illustrate it further.

In the domain of software development, the term Interface is given a concrete definition. On the one hand, the term User Interface (UI) is employed to denote how users interact with the system (Oviatt et al., 2017) (see Section 2.2.10); on the other hand, the term Application Programming Interface (API) refers to the interface between different components of software, such as a client and a server (see Figure 12) (Clark, 2020). This chapter focuses APIs and Figure 13 illustrates the interface between client and server by highlighting key components.

An API defines functionalities that are independent of their respective implementations, thereby ensuring that definitions and implementations can vary without affecting the interface. In the case of client-server communication, protocols are used to define the functionality, with a protocol defining the type and structure of a request (Clark, 2020).

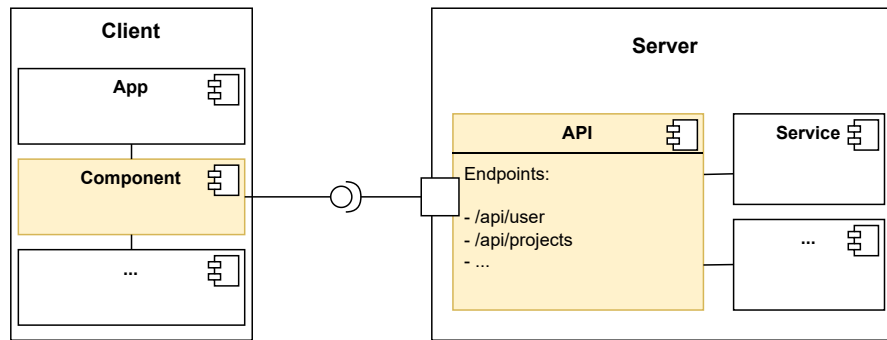


Figure 13: Interface

2.3.2. Hypertext Transfer Protocol – HTTP

The Hypertext Transfer Protocol (HTTP) is a generic, stateless protocol that enables data transfer at the application level. HTTP allows systems to be set up independently of the data to be transferred, as it is the typing and negotiation of the data representation. The World-Wide Web global information initiative uses HTTP since 1990 (Fielding et al., 1999).

HTTP is used in particular in the client-server model (see Figure 12). Clients make requests to a server. In response, the server transmits the requested information to the clients, thus facilitating a data exchange. The following will provide a detailed explanation of the fundamental structure of an HTTP-Request and HTTP-Response.

HTTP-Request

An HTTP request is comprised of three elements. Firstly, the request line, which specifies the HTTP method and the Uniform Resource Locator (URL), thereby delineating the action to be executed and the location to be addressed. Secondly, an HTTP request contains an HTTP header or a set of headers. A header contains, for example, the content type, i.e. how the data is to be interpreted. Additionally, an authorization token may be included, serving to verify the sender's authorization for the requested action. The final element is the request body, which contains the information to be transmitted (Fielding et al., 1999).

Figure 14 illustrates this by showing a POST request for the creation of a project. The example in Figure 14 defines the HTTP method POST in the request line. HTTP methods determine the operation performed on a resource. As illustrated in Figure 15), the four main HTTP methods POST, GET, UPDATE and DELETE can be transferred to the so-called CRUD (Create, Read, Update, and Delete) operations (Battle and Benson, 2008). In summary, the HTTP methods encompass the following functions (in accordance with the CRUD model):

- The **POST** method is used to create a resource.
- The **GET** method is used to get a resource.

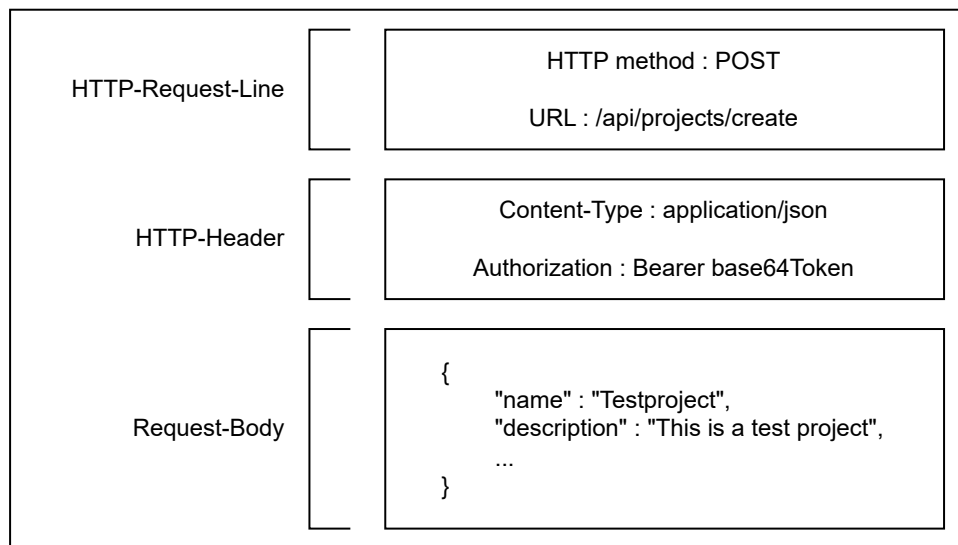
HTTP-Request

Figure 14: HTTP-Request

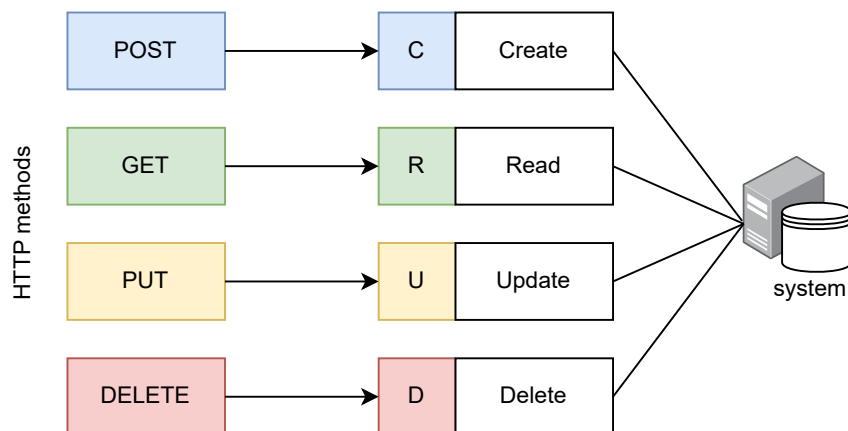
HTTP methods - CRUD

Figure 15: HTTP methods - CRUD

- The **PUT** method is used to update an existing resource.
- The **DELETE** method is used to delete a resource.

HTTP-Response

An HTTP response is analogous to the structure of an HTTP request, with the addition of an HTTP status code. HTTP status codes are three-digit numbers, with the first digit providing information about the type of response:

- **2xx** informs about a successful operation, e.g. 200 (Ok) or 201 (Created)
- **3xx** signals a redirection, e.g. 304 (Not Modified)
- **4xx** describes an error on the client side, e.g. 400 (Bad Request) or 404 (Not Found)
- **5xx** describes an error on the server side, e.g. 500 (Internal Server Error)

Within the numerical range, i.e. the following numbers after the first, various gradations concretize an error. In addition to the HTTP status code, the HTTP response also has a response body. Analogous to the request body, this body contains the data that was requested by a GET request, for example (Battle and Benson, 2008).

2.3.3. RESTful Web Services

RESTful Web Services have gained widespread acceptance across the Web as a simpler alternative to SOAP and WSDL-based Web services. This trend is evidenced by major Web 2.0 service providers such as Yahoo, Google, and Facebook adopting REST while deprecating their SOAP interfaces (Rodriguez, 2008). REST (Representational State Transfer) is an architectural style introduced by Roy Fielding in 2000 that defines constraints for distributed hypermedia systems. To ensure the success of the World Wide Web, Fielding identified several constraints that REST must adhere to (Giessler et al., 2015):

1. Client and Server
2. Statelessness
3. Layered Architecture
4. Caching
5. Code on Demand (optional)
6. Uniform Interface

The uniform interface constraint is particularly important and can be further decomposed into four sub-constraints (Giessler et al., 2015):

1. Identification of resources
2. Manipulation of resources through representations
3. Self-descriptive messages
4. Hypermedia as the engine of application state (HATEOAS)

A service is considered truly RESTful only when it fulfills all of these constraints, with the exception of Code on Demand which is optional (Giessler et al., 2015). One of the fundamental aspects of REST is the separation between resources and their representations. In REST, clients interact with representations of resources rather than with the resources directly. A resource representation typically reflects the current state of a resource and its attributes at the time a client application requests it. These representations can be simple, such as database records mapped to XML tags, or they can reflect complex data models. The key point is that they are mere snapshots in time of the resource state. This decoupling allows the same resource to be represented in different formats based on client needs. For content negotiation, RESTful services utilize MIME types and the HTTP Accept header, allowing clients to choose their preferred data format. Common MIME types used in RESTful services include (Rodriguez, 2008):

- application/json for JSON
- application/xml for XML
- application/xhtml+xml for XHTML

This mechanism minimizes data coupling between the service and its client applications, enabling various clients on different platforms to use the service effectively (Rodriguez, 2008). A key characteristic of RESTful Web Services is the explicit use of HTTP methods as defined in Section 2.3.2. REST leverages these standard HTTP methods to establish a one-to-one mapping with CRUD (Create, Read, Update, Delete) operations, as already illustrated in Figure 15. As a general design principle, RESTful URIs should use nouns rather than verbs, as the HTTP methods (verbs) are already defined by the protocol. The Web service should not define additional verbs or remote procedures like `/adduser` or `/updateuser`. This approach applies to the HTTP request body as well, which should transfer resource state rather than carry the name of a remote method to be invoked. According to Fielding, URIs should be used for the unique identification of resources. In RESTful Web Services, URIs should be intuitive and easy to understand, functioning as self-documenting interfaces that require minimal explanation (Rodriguez, 2008). Best practices for resource identification include (Giessler et al., 2015):

1. URIs should be self-explanatory and intuitive
2. Resources should be addressable by collection URIs
3. Resource identifiers should be difficult to predict for security reasons
4. URIs should not contain verbs as this implies a method-oriented approach

To create intuitive URIs, a directory structure-like approach is recommended. This hierarchical structure, rooted at a single path with branching subpaths, exposes the service's main areas in a predictable manner (Rodriguez, 2008). For example:

```
http://www.myservice.org/discussion/topics/{topic}
```


Additional guidelines for URI structure include:

- Hide server-side technology extensions (.jsp, .php, .asp)
- Keep everything lowercase
- Replace spaces with hyphens or underscores (consistently)
- Avoid query strings when possible
- Provide a default resource instead of 404 errors for partial paths

URIs should remain static even when the resource or service implementation changes, allowing for bookmarking and maintaining relationships between resources independently of their storage representation (Rodriguez, 2008). While HTTP is inherently stateless as mentioned in Section 2.3.2, REST elevates this characteristic to an architectural constraint. Statelessness in REST architecture means that each request from a client to a server must contain all the information needed to understand and complete the request. The server should not store any client context between requests, eliminating the need for session data, cookies, or similar mechanisms for storing session information. This stateless design improves Web service performance and simplifies the implementation of server-side components by removing the need to synchronize session data with external applications. It enables better scalability as requests can be distributed across load-balanced servers without maintaining shared state. In a RESTful Web service, responsibilities are clearly divided between server and client, as you can see in the following based on (Rodriguez, 2008):

The server:

- Generates responses that include links to related resources, allowing applications to navigate between them
- Indicates whether responses are cacheable to improve performance

The client:

- Determines whether to cache resources based on the Cache-Control header
- Uses conditional GET requests with the If-Modified-Since header to avoid unnecessary data transfers
- Sends complete requests that can be processed independently of other requests

This collaboration between client and server is essential for maintaining statelessness in RESTful Web services, resulting in improved performance through bandwidth savings and minimized server-side application state (Rodriguez, 2008).

2.3.4. WebSocket

WebSockets have emerged as a pivotal technology in enabling real-time, bidirectional communication over the web, making them especially well-suited for applications where low latency and continuous data exchange are critical—such as chat applications (Skvorc et al., 2014). Traditional HTTP communication is based on a request-response paradigm, which inherently introduces latency and overhead due to the stateless nature of the protocol. In contrast, WebSockets establish a persistent connection between a client and a server, allowing both parties to send data asynchronously as soon as it becomes available. This persistent, full-duplex connection significantly reduces the need for repeated handshakes and the associated overhead, thereby enhancing the responsiveness and scalability of interactive systems.

Client-Handshake

To start the WebSocket connection, an opening handshake is needed, which is an HTTP upgrade request (Melnikov and Fette, 2011).

```
1 GET /chat HTTP/1.1
2 Host: server.example.com
3 Upgrade: websocket
4 Connection: Upgrade
5 Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
6 Origin: http://example.com
7 Sec-WebSocket-Protocol: chat,
```

Sourcecode 1: Client-Handshake

- **Request-URI: GET /chat**

In order to allow multiple domains to be served from one IP address and an array of WebSocket endpoints to be accessed by a single server, the Request-URI is used to identify the endpoint of the WebSocket connection.

- **Host: server.example.com**

This header field is used so that both the client and the server can confirm that they are communicating with the correct host.

- **Origin: http://example.com**

It communicates the originating source of the script that initiated the WebSocket connection, enabling the server to ascertain the origin of the request. If the server opts not to accept connections from this particular origin, it may reject the connection by issuing an appropriate HTTP error code.

- **Upgrade: websocket**

Signals to the server that the client wishes to change the protocol from HTTP to WebSocket. It acts as an explicit request to "upgrade" the connection, ensuring that

both endpoints understand that the communication will switch from the standard HTTP protocol to the WebSocket protocol.

- **Connection: Upgrade**

Indicates that the client intends to use the current connection for the protocol upgrade. It specifies that the header fields in the request should be interpreted in the context of an upgrade, thereby prompting the server to perform the necessary protocol switch.

- **Sec-WebSocket-Protocol: chat**

This field is utilized by the client to enumerate acceptable subprotocols, which are application-level protocols layered over the WebSocket protocol. In response, the server selects one of the provided protocols—or none—and echoes this choice in its handshake, thereby indicating the protocol it has agreed to use for the communication session.

- **Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==**

To confirm receipt of the client's WebSocket handshake and ensure that the server only accepts legitimate WebSocket connections, the server must demonstrate its acknowledgment of the handshake. This mechanism prevents malicious actors from deceiving the server with crafted packets sent via methods such as XMLHttpRequest or form submissions.

To provide this proof, the server utilizes two key pieces of information to generate a response. The first is the value provided in the Sec-WebSocket-Key header field of the client's handshake. The server takes this base64-encoded string (after removing any extraneous whitespace) and concatenates it with the Globally Unique Identifier (GUID) "258EAF5E-E914-47DA-95CA-C5AB0DC85B11," a string that is unlikely to be used by endpoints unfamiliar with the WebSocket protocol. Next, the server computes a SHA-1 hash of the concatenated string, and then base64-encodes the hash. For example, if the Sec-WebSocket-Key is "dGhlIHNhbXBsZSBub25jZQ==", the concatenated string becomes

"dGhlIHNhbXBsZSBub25jZQ==258EAF5E-E914-47DA-95CA-C5AB0DC85B11".

The server then computes the SHA-1 hash, yielding a 160-bit value, which is subsequently base64-encoded to produce "s3pPLMBiTxaQ9kYGzzhZRbK+xOo=". This resultant value is then returned in the Sec-WebSocket-Accept header field of the server's handshake response.

Server-Handshake

The server's handshake is considerably more straightforward. It begins with an HTTP Status-Line that confirms the success of the handshake—any status code other than 101 indicates that the WebSocket handshake has not been completed and that standard HTTP semantics remain in effect. The Upgrade and Connection headers retain the same functionality as in the client's handshake. Additionally, the server computes the Sec-WebSocket-Accept header by concatenating the client's Sec-WebSocket-Key with

the GUID "258EAF5-E914-47DA-95CA-C5AB0DC85B11", computing the SHA-1 hash of the result to produce a 160-bit value, and then base64-encoding that value to yield "s3pPLMBiTxaQ9kYGzzhZRbK+xOo=" (Melnikov and Fette, 2011).

```

1 HTTP/1.1 101 Switching Protocols
2 Upgrade: websocket
3 Connection: Upgrade
4 Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

```

Sourcecode 2: Server-Handshake

Comparison of WebSocket and HTTP

Qigang and Sun, 2012 conducted an empirical evaluation comparing traditional HTTP with WebSocket by measuring the performance of sorting a five-column, three-row table in phpMyAdmin. They systematically documented the performance metrics for both HTTP requests and WebSocket communications, as displayed in Table 1.

Table 1: Comparison HTTP and WebSocket (Qigang and Sun, 2012)

	Number of packets		Number of bits		Time (seconds)	
	HTTP	WebSocket	HTTP	WebSocket	HTTP	WebSocket
Client to server	83	5	33 662	372	—	—
Server to client	77	8	45 600	7 456	—	—
Total	160	13	79 262	7 828	~2.5	~0.25

The comparative analysis indicates that WebSocket is substantially more efficient than HTTP for the given data transfer scenario. By reducing the number of packets by a factor of approximately 12, lowering the data volume by a factor of 10, and decreasing the transmission time by a factor of 10, WebSocket minimizes overhead and latency. This efficiency gain is particularly beneficial in applications requiring real-time data exchange, where the persistent connection and reduced framing overhead of WebSocket provide clear advantages over traditional HTTP request/response cycles.

2.3.5. JSON

JSON (JavaScript Object Notation) is a data format that is both simple and human-readable. It is based on the data types of the JavaScript programming language and essentially consists of key-value pairs. In addition to simple dictionaries, JSON can contain arrays and atomic types such as numbers and strings. Both the value and the contents of arrays and dictionaries can be JSON, which means that any nesting is possible and the JSON format is completely compositional. This attribute, in addition to its simplicity, is the primary reason why JSON is one of the most popular data exchange formats (Bourhis et al., 2020).

```
1 {  
2   "firstName": "Max",  
3   "lastName": "Mustermann",  
4   "age": "27",  
5   "hobbies" ["reading", "football", "basketball"]  
6 }
```

Sourcecode 3: JSON-Example

Sourcecode 3 illustrates an example of a JSON document. Curly brackets are used to initialize an object. A JSON document is characterized by its opening and closing curly brackets, with the document beginning on line 1 and ending on line 6. Name-value pairs are separated by a colon, with both attribute names and string values enclosed in inverted commas. Lists or arrays are defined by square brackets, as demonstrated in line 5.

2.3.6. JSON Web Token (JWT)

To safeguard data from unauthorized access or manipulation, an application must incorporate an authentication or authorization mechanism. Authentication can be defined as the process of verifying the user's authenticity, for example by requesting a username and password. Authorization, on the other hand, concerns the authorization of a user to access resources (Balzert, 2011, p. 153 seqq.). The implementation of these two mechanisms can be achieved by using JSON Web Token (JWT) authentication and they are used in client-server communication with HTTP, as shown in Figure 14 in the HTTP header (Authorization: Bearer base64Token) (Gunawan and Rahmatulloh, 2019).

A JWT can be defined as an encoded JSON object (see Section 2.3.5) that serves as a signature for communication. This signature enables encryption and/or authentication (Jones et al., 2015).

Structure and content of a JWT

A JWT is constituted of three sections, separated by commas, with each section representing a base64-encoded JSON object. The following components are integral to a JWT (Gunawan and Rahmatulloh, 2019):

- **Header:** Contains general information, including the algorithm employed to generate the signature.
- **Payload:** Defines the validity period of the token, among other parameters.
- **Signature:** Includes customized information, referred to as claims, which can be incorporated into the payload. The signature serves to protect the token against manipulation.

As illustrated in Figure 16, a snippet of the jwt.io website from the Okta company provides an example of the JWT generation.¹ The header, payload, and signature on the right-

¹<https://jwt.io/>

hand side are encoded in the token on the left-hand side. Color coding illustrates the affiliation. The combo box above, in conjunction with the content of the header, indicates the algorithm employed in the creation of the JWT.²

Algorithm HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Ik1heE11c3Rlcm1hbm4iLCJwYXNzd29yZCI6IiRlc3QxMjM0In0.cF9xNmce8-aPsY4Trw0Icv0WcqQDAmyLoUqvfd1W0fs
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "username": "MaxMustermann",
  "password": "Test1234"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Figure 16: Example JWT: Encoding/Decoding

Authentication process

In the context of JWTs, the process of creation is often inextricably linked to the act of user login, typically initiated through the client application. The user's login credentials, such as their username and password, are transmitted to the server for validation. This validation process involves verifying the existence of the specified account. If the login credentials are found to be valid, the authentication process is deemed successful. Consequently, the server generates a JWT based on the received data. This JWT is then dispatched as a component of the response to the user's login data. The client then saves the JWT and sets it in the HTTP header for each subsequent request (see Figure 14). The server can then validate whether the sending client is authorized for the request by examining the JWT contained in the HTTP header. If the JWT is invalid, the request is rejected (Balzert, 2011, p. 153 seqq.) (Gunawan and Rahmatulloh, 2019). Figure 17 illustrates this using a sequence diagram.

²<https://jwt.io/>

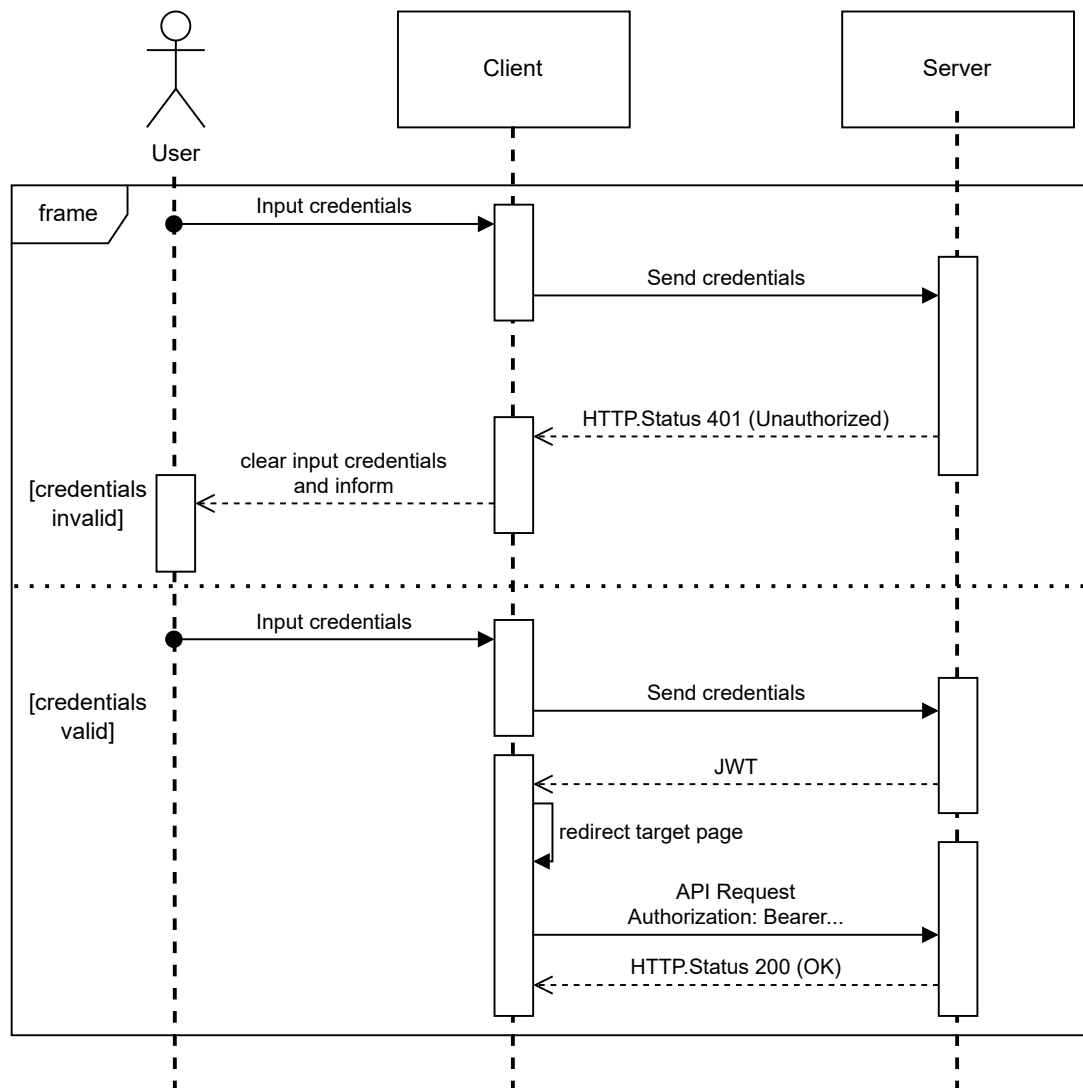


Figure 17: Authentication process

3. Project Concept

3.1. Project Description

The goal of this project was to develop a multimodal interface for LLMs that allows users to upload, edit and create documents, in particular PDFs, with the support of an AI-powered assistant. The interface aims to enable the user to highlight different elements - like text, images or tables - in the PDF, which can then be further processed by the integrated AI assistant.

TextVision is a web-application that facilitates collaborative work by enabling users to edit and refine documents together. A key aspect of TextVision are its integrated PDF viewer and editor components, which allow users to directly interact with their documents. The editor enables text changes, annotations and structured prompt-based interactions. Users can create documents via the editor function, which allows the input and editing of user-created text or speech input. Included within the application is an LLM-based chatbot that acts as an AI assistant for document analysis and editing. Users can ask questions, highlight sections for prompts, and perform various text editing tasks. To minimize repetitive tasks, frequently used prompts can be saved as buttons, allowing users to highlight text and perform predefined tasks efficiently. The chatbot provides cross-language support, explains technical terms, and offers context-sensitive suggestions. Additionally, it recommends or adjusts prompts within the chat interface to enhance editing results. The tool also offers text-to-speech functionality, which converts text into spoken language, and speech-to-text transcription, which allows users to dictate content directly. Image import is also supported, allowing visual elements to be integrated into the document workflow or for the chatbot to work with contents displayed in a provided image. To further enhance its capabilities, the application includes a prompt designer that allows users to create, refine, and track the progress of custom AI prompts, ensuring flexibility and customization.

The integrated interactive chat system facilitates direct communication with the chatbot. To minimize the repetitive formulation of frequently used prompts, the application supports storing prompts as buttons. Users are able to mark text and execute tasks using these buttons. Furthermore, TextVision also supports a voice command functionality. For example, the keyword "TextVision Summarize" summarizes the currently marked content in the PDF and extracts the response containing the finished summarization from the chat. This feature further tries to enhance the usability of TextVision by accommodating different input modalities.

3.2. Project Management and Tools

This following section will focus on the implementation of project management and tools for the development of the application. Furthermore, the chapter will contain information about the expert roles. In the end, the procedure will be evaluated to identify problems and improvements.

3.2.1. Tools

Depending on the tasks, various tools were used. The applications can be found compiled in Figure 18.

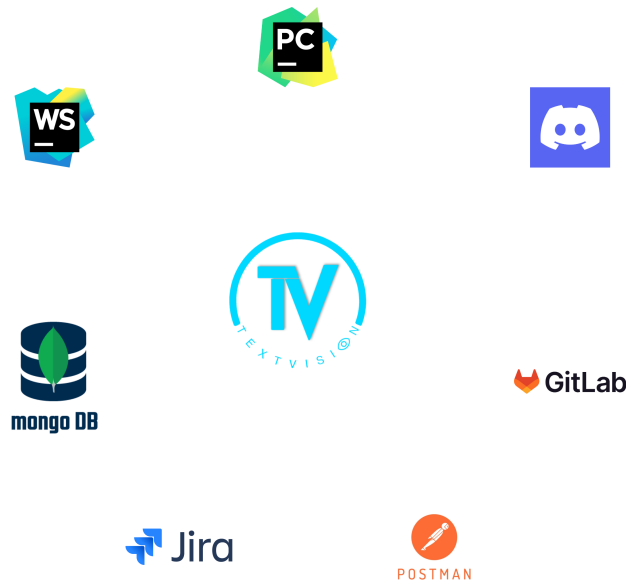


Figure 18: Important Tools

For the development of TextVision, two primary development environments were used: PyCharm for the Python code and WebStorm for the Node.js backend and frontend. Additionally, MongoDB Compass and Postman were utilized during development. MongoDB Compass was used for managing and visualizing the database, while Postman was used to test the backend code, enabling the testing of various endpoints independently of the frontend application.

The project group organized itself using Discord, Jira, and GitLab. For internal team communication, the Discord application was used. A dedicated server with voice and chat channels was created to facilitate communication among team members.

For workflow management, the Atlassian Jira tool was used. Tasks were assigned to each participant, with progress transparently documented. Jira supported task planning and implementation following the SCRUM framework, which is discussed in more detail in Chapter 3.2.2.

For version control, the GitLab platform provided by DFKI was used. GitLab offers essential functionalities for collaborative work on a shared codebase.

3.2.2. Team Organization and Workflow

There were clear responsibilities and roles within the group. The basic concept was the division into three broad areas of responsibility: frontend, backend and organization. The first step was for each participant to choose whether they wanted to join the frontend or

the backend team. This was based on individual skills and preferences. Each team had a technical lead who was responsible for planning and distributing tasks. This person also acted as a point of contact for technical questions. Everyone in the project group had to be involved in either the frontend or the backend. As is common in software development, the frontend team was responsible for the user interface and the connection of backend functionality. Within the backend there were different areas of focus due to the split between the two backends. However, the entire team was responsible for the structure and implementation of the backend functionality.

In addition, a group of people were selected to maintain an overview of the project. This group included both technical leads and a project group lead responsible for project management. Together, these three individuals were responsible for controlling and monitoring the development phase.

This table (2) shows the different assigned roles:

Table 2: Roles and Responsibilities of Team Members

Person	Roles
Elias Scharlach	Frontend Developer & Lead
Jannik Podszun	Frontend Developer & User Study
Melis Aslan	Frontend Developer & User Study
Leon Selzer	Frontend Developer & UI/UX Expert
Yukun Wu	Frontend Developer & Voice Expert
Maximilian Bosse	Backend Developer & Lead
Philipp Olschewski	Backend Developer & Project Manager
Marlon Hinz	Backend Developer & AI Expert
Daniel Ehlers	Backend Developer & Fine-Tuning & Documentation

During the implementation of TextVision, the group followed the agile development method "SCRUM". Each participant was largely familiar with SCRUM-based development, which made it possible to integrate SCRUM and directly into the project structure. Due to the group size of only 9 participants, the SCRUM structure was applied to the project in a minimized form (see Figure 19).

Using Jira, a product backlog was created and continuously updated with new issues. Each sprint was planned and filled from this backlog during sprint planning.

The leads identified the highest priority tasks and added them to the sprint backlog. Each task was estimated in terms of effort and assigned its own priority. This task was handled by the organizational team and was not discussed again with the group in a separate meeting. Normally the group estimates the effort internally during planning, but in this case the group decided on the estimate with the support of the two most experienced leads. Another deviation from standard SCRUM was that team weekly meetings were held instead of daily meetings. Each internal team could conduct sprint planning during the weekly slot. During this slot, people presented the current status of their tickets and could ask questions about implementation. A sprint usually lasted two weeks and tasks

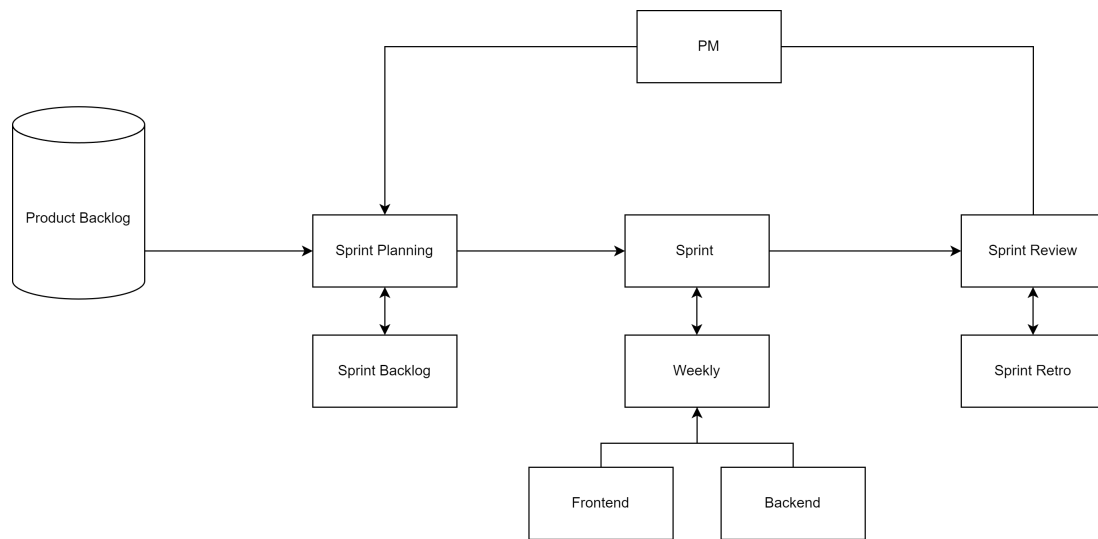


Figure 19: SCRUM in PG AIM

were planned accordingly. For sprint planning, approximately 13 story points per person were allocated, corresponding to the estimated effort of the tasks.

The frontend and backend teams met only once every two weeks for an internal sprint retrospective/review. During this meeting, the progress of the project was discussed and issues could be raised anonymously in the sprint retrospective. Any issues raised were then discussed with the team and efforts were made to resolve them. Feedback from the team was taken into account by the organizational team for the next sprint. Each meeting was documented to ensure that decisions remained traceable and that absent members could check on progress. Looking back, we would choose to work with SCRUM again, as the agile approach gave us the flexibility to include tasks or focus on specific aspects of development as the situation required. Through planning and retrospectives, all participants were involved and able to contribute their expertise.

However, some critical issues were raised. At times, members had little transparency about the work of other subteams. Midway through the project, it was decided that in weeks when there were no weekly meetings, a representative from each team would attend the other team's meeting. This ensured that input from both teams was considered and specific issues could be addressed directly.

3.3. Scenario-Based Exploration of Key Use Cases & Features

The TextVision platform is designed to address various real-world scenarios by streamlining complex document workflows, offering an integrated environment that enhances productivity and efficiency, particularly in research settings. Due to the rise of Big Data, the volume of available information continues to grow (N. Khan et al., 2014). Consequently, researchers working in e.g. scientific or technical domains also have to deal with a large amount of information, requiring tools like TextVision that try to enhance the productivity and efficiency of the research and document creation process.

Figure 20 provides an overview of the core interactions within TextVision. The PDF Viewer allows users to upload, browse, and highlight text passages. Based on the highlighted text passage, the user also has the ability to directly ask the AI about it or add it as a note for later use in the AI Assistant. The AI Assistant (Chatbot) processes user queries, offers prompt recommendations (see Section 3.5), and enables the integration of generated responses into the Document Editor, where users can manually edit and format their work before exporting it as a PDF. Additionally, the PromptDesigner (see Section 3.5) helps users refine and optimize their prompts through an iterative evaluation process. This will enable users to create and save an optimized prompt for their use cases, which they can reuse in the AI Assistant at any time in the future.

With this foundational understanding of the system's core components and their interactions, the following subsections will explore some of the use cases and features, demonstrating how TextVision supports researchers in real-world scenarios. The overview provided is not an exact, fully detailed representation but rather a simplified abstraction to maintain clarity and avoid unnecessary complexity. A more comprehensive explanation of the entire concept will be presented in the course of the documentation.

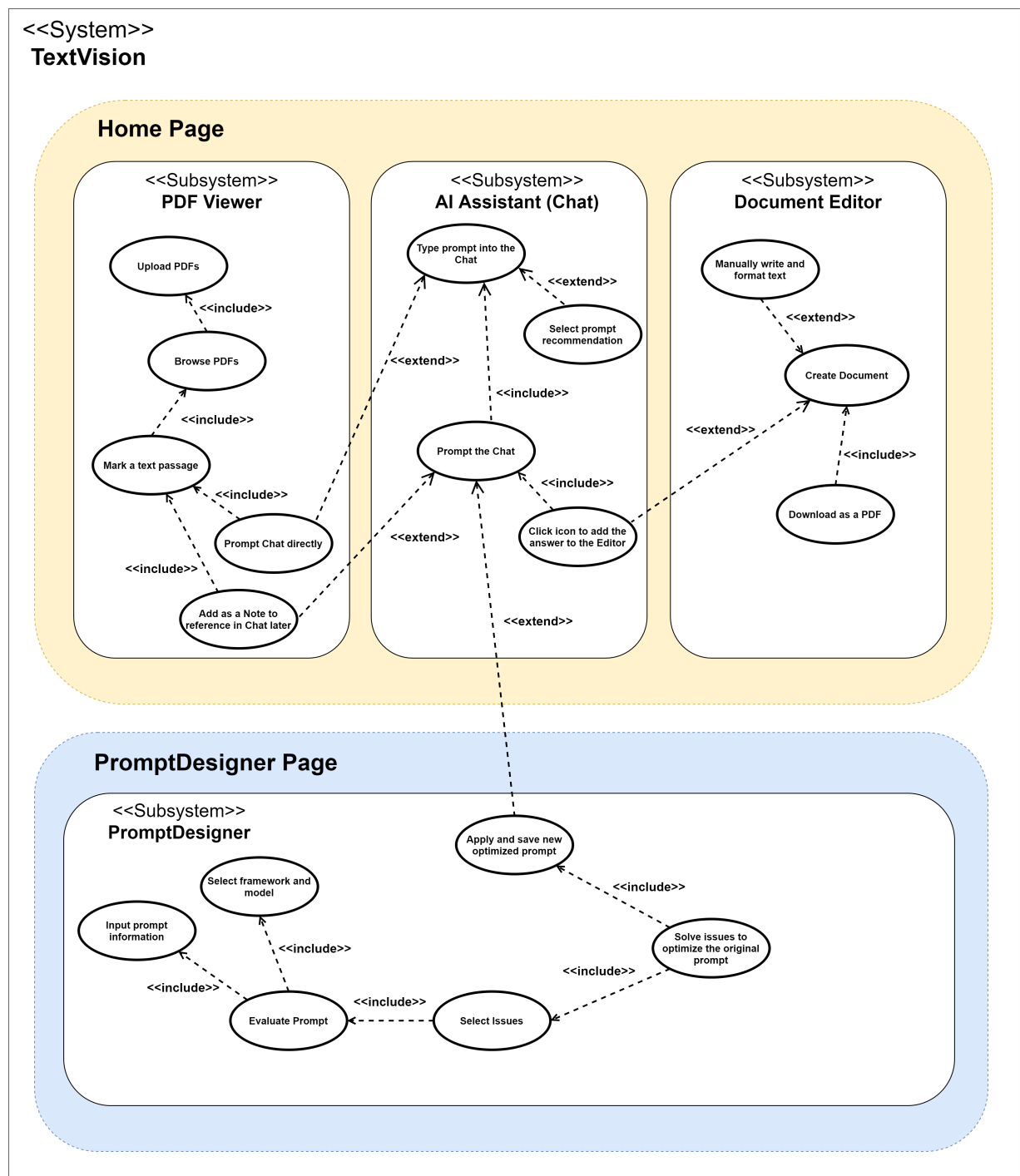


Figure 20: Illustration of the most relevant interactions between TextVision's core components (PDF Viewer, AI Assistant, Document Editor, and PromptDesigner).

3.3.1. Reviewing and Summarizing Research Papers

A challenge for researchers could be reviewing large numbers of academic papers, extracting important insights while managing time constraints. TextVision aims to provide a streamlined solution to this problem by allowing users to upload and interact with documents directly within the platform. Figure 21 showcases a simplified interaction with our tool, illustrating how researchers can efficiently engage with documents and extract relevant information.

- **Scenario:** A researcher uploads a set of PDF articles related to a specific topic. Using the integrated PDF viewer, they can browse the uploaded documents and leverage the AI assistant to improve the efficiency in understanding the topic.
- **Interaction:** The user sets a focus by highlighting a specific text passage, such as a complex paragraph in the discussion section, and asks the AI based on this to generate a simplified summary or explanation.
- **Outcome:** This aims to reduce cognitive load, accelerate comprehension, and allow the researcher to focus on synthesizing insights rather than trying to understand dense text.

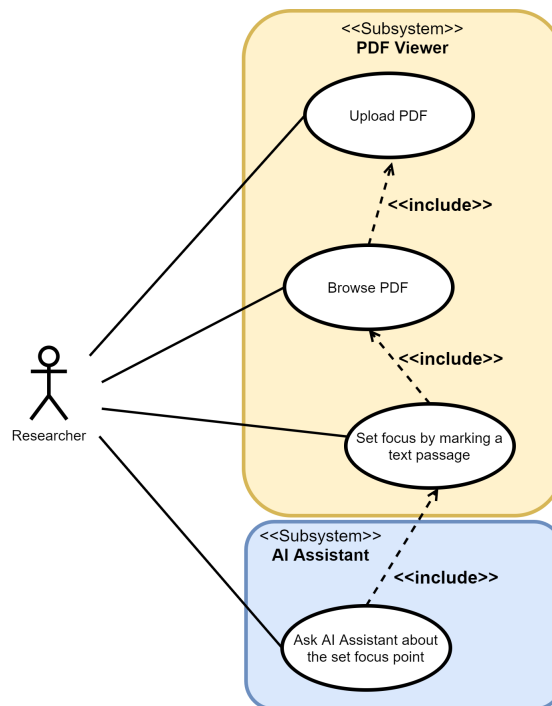


Figure 21: Illustration of how a researcher would generally utilize TextVision to enhance their productivity.

3.3.2. Designing and optimizing research prompts

In research, choosing an optimal heuristic to search for findings is one of the central aspects and part of the **Search Triangle** of the research process (Gusenbauer & Haddaway, 2021). TextVision’s prompting concept (see Section 3.5) tries to enhance this aspect by assisting researchers in finding an optimal prompt as well as creating and optimizing their prompts for specific research tasks. Figure 22 provides a simplified illustration of TextVision’s prompting process.

- **Scenario:** A researcher is conducting a systematic review on the impact of climate change on society. They need to extract specific data from multiple research papers. As they interact with the chat, they receive prompt recommendations based on their current input and the content of the uploaded documents.
- **Interaction:** As the user begins typing a query in the chat, the Prompt Recommendations suggest tailored prompts that are contextually relevant to their current research focus. If the user accepts a suggestion, they can use it immediately. Alternatively, they can optimize the prompt or create a completely new one, in the PromptDesigner window to ensure it aligns with their needs.
- **Outcome:** This process aims to streamline the research workflow by helping researchers create more accurate prompts with less effort. The ability to reuse and optimize prompts ensures that researchers can maintain consistency and efficiency across different phases of their work. These features give researchers more control over the information retrieval process, which can improve the speed and quality of their research.

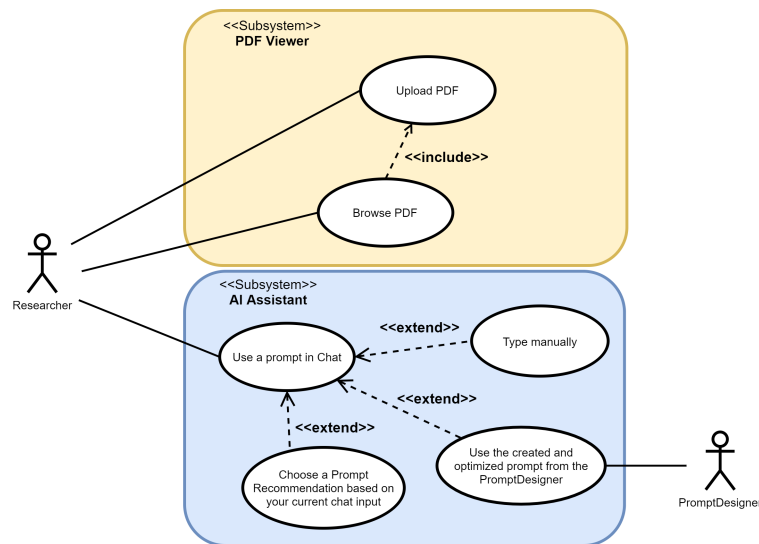


Figure 22: Illustration of the prompting process in TextVision.

3.3.3. Cross-referencing and contextualizing information

Researchers could also encounter situations where they must cross-reference information from multiple sources or require context to understand specific findings better. TextVision aims to facilitate this process by enabling the ability to switch between viewing different documents and integrating context-aware AI assistance. Figure 23 illustrates the general workflow when working with multiple uploaded PDFs.

- **Scenario:** While working on a literature review, a user identifies two papers with differing conclusions about a similar hypothesis. By uploading both documents, they can use TextVision to cross-reference specific sections and query the AI-Assistant for contextual explanations or summaries of key differences.
- **Interaction:** The user highlights relevant paragraphs in each paper and prompts the AI assistant to compare methodologies or results, providing a synthesized analysis.
- **Outcome:** This aims to the researcher in uncovering insights and enhancing their ability to draw informed conclusions for their work.

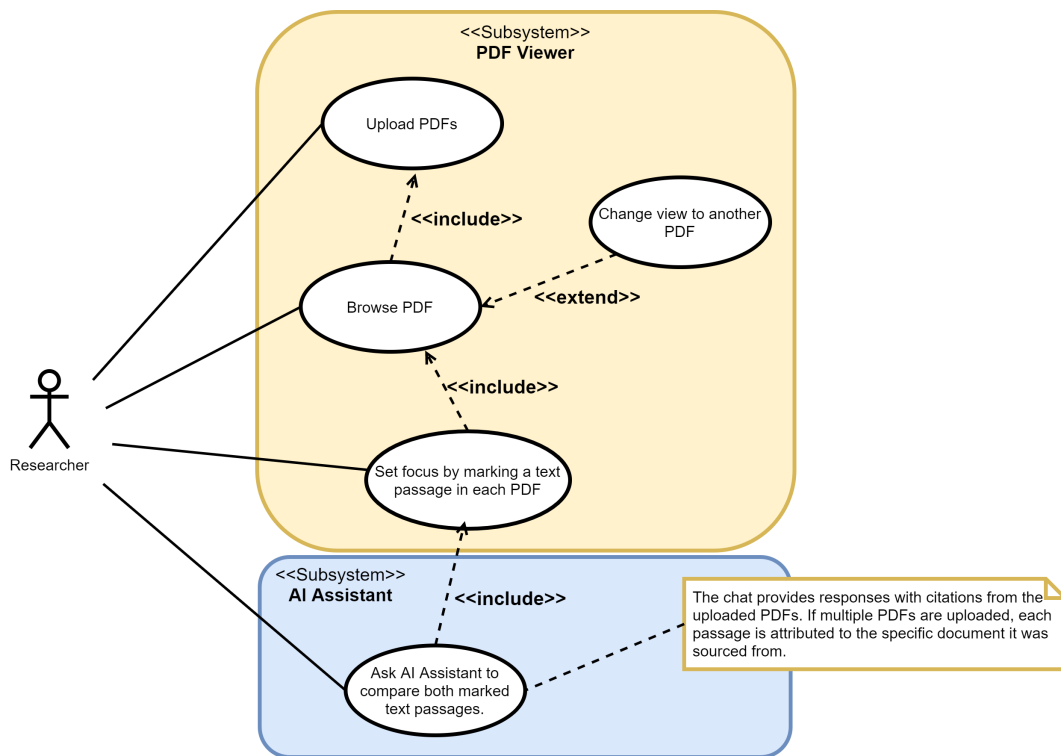


Figure 23: Illustration of cross-referencing and contextualization when working with multiple uploaded PDFs.

3.4. User Interface

TextVision’s user interface integrates a PDF editor and chat interface on a single page to streamline the workflow and minimize unnecessary effort. While the design has evolved over time, the initial vision for the interface can be seen in the earliest mockup of the product, depicted in Figure 24.

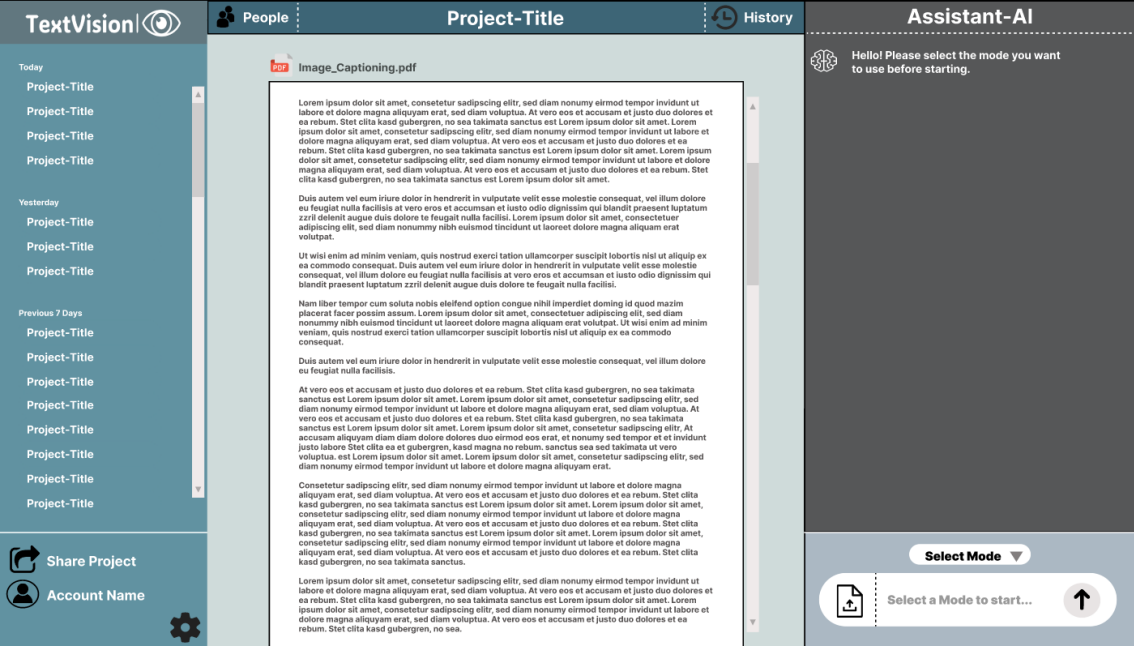


Figure 24: Initial Mockup of TextVision

The goal of this project is for users to complete tasks with as little friction as possible, while for the developers, the aim is to create a smooth and intuitive experience. After creating an account, users are directed to the Project Center, where they can set up workspaces that can either be shared with selected users or used individually. Once a workspace is opened, the user is directed to the main component called Home, where they can work on their documents. To increase efficiency, the design follows a single-page structure where the PDF Viewer, Editor, and Chat component are simultaneously visible. This setup allows users to upload PDFs, discuss them with the AI Assistant, and incorporate responses directly into the editor without having to switch between multiple tabs. This approach aims to improve the user experience by ensuring that users can focus on creating documents without interruption.

Clicking on the TextVision logo reveals a Sidebar for navigation. The sidebar contains links to Home, PromptDesigner, FAQ, Profile, and Debug. PromptDesigner follows a structured layout to guide users systematically. The Prompt Configuration pane allows users to modify their prompts by selecting options from a combo box. If a prompt has not yet been created, users must first name it in the field provided. In the prompt field, they

enter a statement, question, or instruction for the AI to respond to, ensuring clarity and specificity. In the Goal field, users define the main purpose of the prompt, which helps to provide context for the AI's response. The Description field allows users to add additional details to refine the output. Once the prompt is defined, users move to the scoring panel, where they select a GPT model, choose a framework, and adjust the creativity level using the temperature slider. These settings allow for fine-tuned responses tailored to the user's needs. The final step in the process is scoring the prompt, which produces a result that includes a star rating, identified issues, and suggested improvements. Users can then apply the recommended changes to tweak their prompt for greater accuracy and relevance.

Understanding how users interact with prompts is essential to optimizing their experience. The PromptDesigner is a dedicated interface that helps users refine their prompts by suggesting improvements and allowing them to apply changes seamlessly. If needed, users can clear the optimization history. The workflow follows a structured process: users begin by selecting or defining a summarization and evaluation prompt. Next, they configure the evaluation parameters and click the Evaluate button to receive feedback. The PromptDesigner then presents suggested improvements for users to review and apply before finalizing their prompt. This iterative approach ensures that prompts are fine-tuned for better results. To support a clear and efficient user experience, the interface follows a structured design. Important elements, such as action buttons and rating indicators, are visually distinct, while sections are clearly labeled and logically grouped to facilitate navigation. Accessibility features such as keyboard navigation, tooltips, and information icons provide additional guidance for users.

For users seeking assistance, the FAQ page serves as a structured knowledge base. It is divided into four categories, each represented by a button with an icon, and includes a search bar for quick access to relevant topics. As users type a query, up to five suggested topics appear in a dropdown menu that can be navigated using either the keyboard or mouse. Selecting a suggestion directs the user to the corresponding category. Each FAQ category includes a brief description of its contents. Questions are presented in a collapsible format, ensuring that only relevant answers are displayed when selected. In addition, a Contact Us button allows users to submit inquiries via a modal form where they can specify a topic, enter their email, and submit their request for further assistance.

In general, navigation within the platform is designed for efficiency. The sidebar provides access to additional sections such as Profile and Debug, both of which can be expanded as toggleable sections. In the bottom left corner, an Exit Project button allows users to return to the Project Center, ensuring a smooth transition between different functionalities.

3.5. Integration of LLMs & Prompting Concept

As discussed in the introduction, there is a growing trend of users engaging with LLMs without a comprehensive understanding of their underlying mechanisms, which often results in the use of suboptimal prompts (Zamfirescu-Pereira et al., 2023). To address this issue, various strategies have been proposed, including active interaction with LLMs (Mas-

son et al., 2024), the use of predefined prompts, and the development of applications that enhance user-generated prompts (E. Jiang et al., 2022; Sučík et al., 2023).

The initial approach of integrating LLMs involved leveraging them to directly manipulate documents to generate improved versions. However, since the decision had already been made to utilize PDFs—which are designed to maintain stability and consistent appearance across different devices, operating systems, and software (“Document management — Portable document format — Part 1: PDF 1.7”, 2008)—this approach was deemed incompatible. Consequently, the focus shifted to developing a tool that interactively refines user prompts and provides constructive feedback for their improvement (see Section 5.8). The PromptDesigner was developed to enable users to input a prompt and receive feedback on its deficiencies, with the option to automatically incorporate this feedback to produce an enhanced version. Additionally, a prompt suggestions feature (hereafter referred to interchangeably as prompt suggestions/recommendations) was integrated. Specifically, when a user begins typing after a brief period of inactivity, three prompt recommendations are displayed above the input field. This feature is designed to provide users with an initial framework or guidance when they have a general topic for a prompt but lack a precise formulation. Importantly, it leverages the text in the input field in conjunction with the content of PDFs and the chat history to generate contextually relevant and appropriate prompt suggestions.

3.6. System Architecture

As illustrated in Figure 25, the system architecture of TextVision follows a modular, component-based design approach. The architecture comprises three main components: a frontend, a JavaScript backend, and a Python AI backend. These components function together to provide the application’s capabilities.

The frontend utilizes React as the framework, Vite as the build tool, and Bootstrap for UI components. This interface manages user interactions by collecting data and displaying results. Communication with the JavaScript backend occurs via WebSockets for bidirectional real-time communication and Axios for REST HTTP requests.

The JavaScript backend employs Node.js with Express.js as the web application framework. This layer processes frontend requests, manages MongoDB data storage, and coordinates communication with AI services. MongoDB stores user data, conversation histories, and application states, serving as the persistence layer.

Communication between the JavaScript backend and Python AI backend is established through REST API calls, separating general application logic from AI processing. The Python AI backend uses FastAPI with Uvicorn as the ASGI server for efficient request handling. This component manages AI operations including prompt processing, context management, and model interaction.

The Python AI backend interfaces with various Large Language Models (LLMs) via API integrations, supporting both OpenAI models and fine-tuned alternatives. This architecture enables TextVision to utilize JavaScript for web functionality and Python for AI capabilities while maintaining distinct component interfaces.

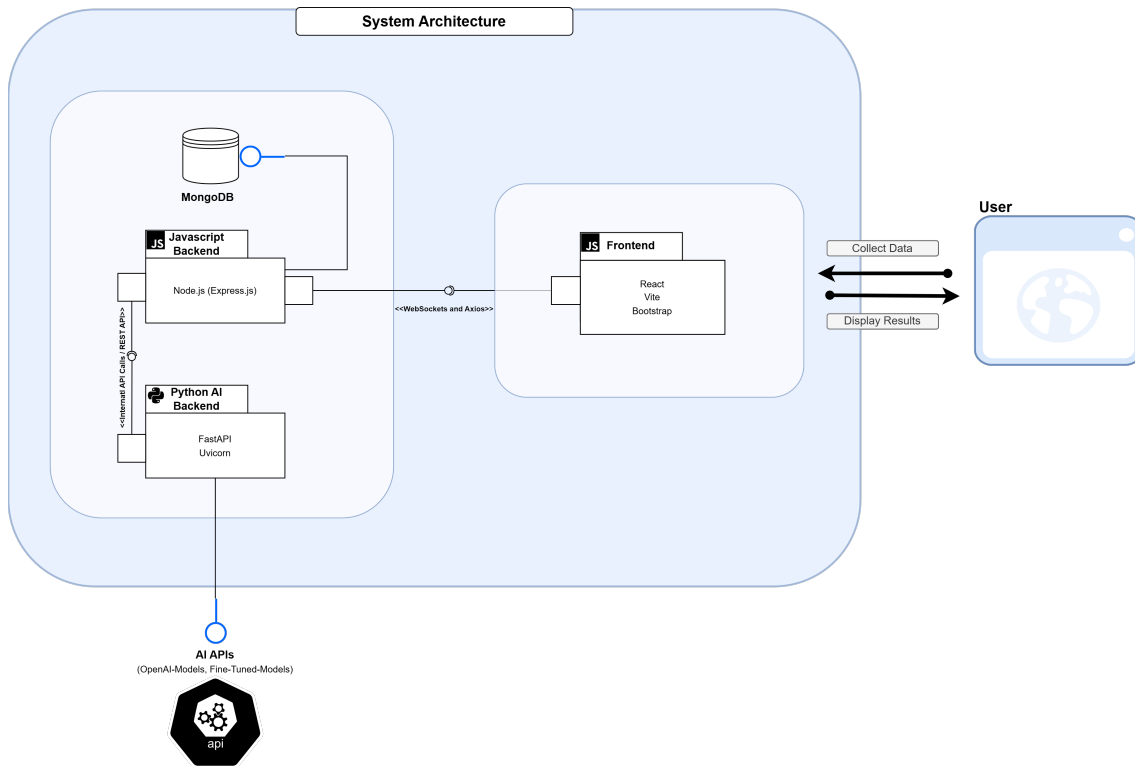


Figure 25: System architecture overview of TextVision

3.6.1. Frontend

The frontend of the system serves as a web server, providing resources to the browser while accessing resources from the backend server. It plays a crucial role in enabling user interaction with the application by rendering the user interface and facilitating communication with the backend for data retrieval and updates.

The frontend of the system is developed using a technology stack that includes **Node.js**, **React**, **Vite**, and **JavaScript**. Each of these technologies contributes to the performance and scalability of the frontend:

- **Node.js:** While primarily used as a backend runtime, Node.js also plays a role in frontend development by enabling server-side rendering and simplifying the build process for JavaScript applications³.
- **React:** A component-based JavaScript library that provides a declarative approach to building dynamic user interfaces; in addition, React's virtual DOM mechanism increases rendering efficiency, resulting in improved performance and responsiveness⁴.

³Is Node.js frontend or backend? A comprehensive guide. (n.d.). Retrieved February 26, 2025, from <https://amorser.com/insights/is-nodejs-frontend-or-backend-a-comprehensive-guide>

⁴Understanding virtual DOM in React. (2024, September 11). Refine. Retrieved February 27, 2025, from

- Vite: A build tool that streamlines frontend development by providing fast module hot loading and an efficient bundling process. Vite significantly improves the development experience by reducing build times and ensuring a seamless workflow⁵.
- JavaScript: The core programming language used to build the frontend, allowing for interactive elements, client-side logic, and asynchronous communication with the backend server.

As shown in Figure 25, the frontend interacts with the backend through defined API endpoints, retrieving and sending data as needed. This separation between frontend and backend ensures single responsibility, modularity and maintainability. The use of modern web technologies enhances performance and supports scalability, making the system capable of handling dynamic user interactions.

While the frontend uses various tools and frameworks, specific component libraries are not discussed in this chapter. These details will be covered in separate frontend sections to provide a more comprehensive understanding of the component structure and styling methodologies used in the project.

3.6.2. Backend

The purpose of the backend is to provide a centralized server that handles incoming requests from the frontend. This includes handling logins and authentication and storing user data as well as processing prompts and forwarding them to the AI backend to then send the AI generated responses back to the frontend. The backend uses a technology stack composed of node.js, express.js, axios and MongoDB, among others. Each of these dependencies contributes towards the backend's purpose of handling incoming and outgoing requests as well as managing the storage:

- Node.js: Similar to the frontend, node functions as the runtime environment, simplifying the build process and providing the node package manager.
- Express: A framework that provides functionalities for an HTTP routing system.
- MongoDB: A NoSQL database that is responsible for storing data related to users, workspaces, model references and prompting frameworks.

As shown in Figure 25, the backend interacts with both, the frontend and AI backend, handling incoming and outgoing requests. It is also connected to the aforementioned MongoDB, thus all functionalities that involve any form of persistent storage, are also handled through the backend.

<https://refine.dev/blog/react-virtual-dom/>

⁵Dissanayake, N. (2024, December 30). Vite.js: Build faster Frontends — Syncfusion blogs. Syncfusion. Retrieved February 27, 2025, from <https://www.syncfusion.com/blogs/post/vitejs-faster-frontend-development>

3.6.3. AI-Backend

The AI backend is a key component within the TextVision system architecture, serving as a bridge between the application logic and various LLMs. Implemented in Python, this backend component leverages the language's extensive ecosystem for artificial intelligence and machine learning applications. The choice of Python was deliberate, given its widespread adoption in the AI community, comprehensive scientific computing libraries, and efficient handling of complex mathematical operations essential for AI processing.

Within the overall system architecture, the AI backend operates as a specialized service layer, receiving requests from the main JavaScript backend via a REST API interface. This separation of concerns allows for independent scaling and maintenance of AI-specific functionality while maintaining system cohesion. The backend implements a modular architecture consisting of several key components that handle different aspects of AI processing, from model interaction to document analysis.

The core functionality of the system includes AI model interaction handling, document processing, and vector-based search capabilities. It uses a modern web framework for API implementation, ensuring efficient request handling and response generation. The architecture supports asynchronous operations, allowing efficient processing of multiple requests while maintaining system responsiveness. The implementation of vector storage allows for sophisticated document search and analysis capabilities, enhancing the system's ability to process and analyze textual data.

Security concerns are addressed through the implementation of authentication mechanisms and secure communication protocols. The system supports multiple AI models through a flexible backend architecture that allows for the integration of OpenAI models as well as custom fine-tuned alternatives. This flexibility ensures that the system can adapt to evolving AI technologies and specific application requirements.

The AI backend architecture emphasizes modularity and extensibility, implementing established design patterns that facilitate system maintenance and future enhancements. Error handling and logging mechanisms are integrated throughout the system, ensuring operational reliability and facilitating debugging. The design of the backend allows for horizontal scaling, making it suitable for use in various operational scenarios.

This component plays a substantial role in TextVision's overall architecture, processing AI-related tasks through well-defined REST API endpoints that establish standardized communication patterns with the JavaScript backend. These endpoints provide a consistent interface for model interactions, document processing, and vector-based searches, ensuring reliable data exchange between system components. The following chapters will examine each aspect of the AI backend in detail, providing comprehensive insights into its implementation and capabilities.

3.7. Documentation JSDoc

JSDoc adds structured comments to the code. These comments increase code readability, improve maintainability, and facilitate automated documentation generation. JSDoc annotations allow developers to specify parameter types, return values, function descriptions,

and examples, making JavaScript codebases easier to understand and use.

JSDoc was chosen to document the codebase because of its ability to:

- Improve **collaboration** among team members by providing clear explanations of function inputs, outputs, and behavior.
- Improve **code understanding** for future developers, reducing onboarding time.
- Enable **automated documentation generation**, ensuring up-to-date documentation with minimal effort.

Integration Process

The example used here is one class from the frontend and one class from the backend. JSDoc has been integrated into both the frontend (PDFViewer.jsx) and backend (chatRepository.js) files to provide structured documentation for functions, variables, and components. The following annotations have been used:

Frontend Example: PDFViewer.jsx

Component Documentation

```
1 /**
2  * PDFViewer Component
3  *
4  * The PDFViewer component serves as a comprehensive solution
5  *   for viewing,
6  * uploading, and interacting with PDF documents. It offers a
7  *   user-friendly
8  * interface that allows users to seamlessly navigate through
9  *   PDF files,
10  * zoom in and out, and search for specific text within
11  *   documents.
12  *
13  * @component
14  */
```

Function Documentation

```
1 /**
2  * Fetches PDF files when the component mounts.
3  * @returns {void}
4  */
5 useEffect(() => {
6   refetchPdfs();
7 })
```

```
7 }, []);
```

Variable Documentation

```
1 /** @type {Array} localPdfs - Stores locally uploaded PDFs */  
2 const [localPdfs, setLocalPdfs] = useState([]);
```


Parameter and Return Type Documentation

```
1  /**
2   * Handles file input change and uploads the selected PDF
3     file.
4   * @param {Event} event - File input change event.
5   */
6   const onFileChange = (event) => {
7     const uploadedFiles = Array.from(event.target.files);
8   };
```

Backend Example: chatRepository.js

Function Documentation

```
1     /**
2     * Saves a chat message to a user's chat history.
3     * @param {string} workSpaceId - The ID of the workspace the
4       message is associated with.
5     * @param {string} userId - The unique identifier of the user
6       .
7     * @param {object} message - The message object containing
8       the content of the chat.
9     * @param {Date} time_sent - The timestamp of when the
10      message was sent.
11    * @param {boolean} isServer - Flag indicating if the message
12      is sent by the server.
13    * @returns {Promise<object>} - An object indicating the
14      success status and message of the operation.
15    * @throws {Error} - Throws an error if the userId format is
16      invalid or if the user is not found.
17    */
```

Error Handling Example

```
1     /**
2     * Retrieves the chat history of a user.
3     * @param {string} userId - The unique identifier of the user
4       .
5     * @returns {Promise<object>} - An object containing the
6       success status, a message, and the chat history.
7     * @throws {Error} - Throws an error if the userId format is
8       invalid.
9     */
10    retrieveChatHistory = (userId) => {
```

```
8 | if (!ObjectId.isValid(userId)) throw new Error("Invalid_
9 |   userId_format");
   | };
```

By choosing to integrate JSDoc into the project, the main idea is to significantly improve documentation quality, making it easier for group members, developers, and outsiders to understand, maintain, and extend the codebase.

4. Interfaces (API)

4.1. Data-exchange between Frontend and Backend

As described in Section 2.3.1, an interface enables communication between two applications. The server provides an endpoint that the client can use, derived from the use cases in Section 3.3. Tables grouped according to the application are employed below for the interface description.

It is important to note that the responses listed in the tables correspond to successful and valid requests. Status codes are not explicitly included in the responses (e. g., 200 for success, 400 for invalid requests, and 500 for server errors).

4.1.1. Authentication API

As illustrated in Table 3, the Authentication API encompasses five endpoints, extending beyond the scope of authentication to include fundamental functionalities such as the creation and dissolution of user accounts.

Table 3: Endpoints of the Authentication API

Functionality	Request-Line	Request	Response
Register	POST /api/auth/sign-up	{userName, pass-word}	{accessToken, refreshToken, expiresIn}
Login	POST /api/auth/login	{userName, pass-word}	{accessToken, refreshToken, expiresIn}
Refresh-Token	POST /api/auth/refresh-token	{accessToken, refreshToken}	{accessToken, refreshToken, expiresIn}
Logout	POST /api/auth/sign-out	Authorization	
Delete Account	POST /api/auth/deleteUser	Authorization	

The **Register** endpoint is employed to create a user account. An HTTP POST request to the path `/api/auth/sign-up` is utilized for the transmission of the desired user name and password. If both values are found to be valid, the client is provided with the JWT in response, in addition to the time at which it expires and another token that is required for the update of the authentication. According to the returned data, it is discernible that a valid registration is automatically linked to authentication.

If a user account already exists, the **Login** endpoint may be utilized for authentication purposes. The user name and password of the user are transmitted via an HTTP POST request (`/api/auth/login`). Following the successful verification of these credentials by the server, the client is furnished with the same data as was received during the initial registration process.

In the event that the user has already been authenticated and is therefore logged in, it is possible that the JWT has expired and a new token must be requested. The **refresh token** is required for the request. An HTTP POST request is used to transmit the expired JWT and the refresh token supplied during the authentication process (`/api/auth/refresh-token`). The server responds with updated tokens.

The user can **log out** of the server actively by means of an HTTP POST request. In order to accomplish this, the user is required to address the path `/api/auth/sign-out`, whereupon the server will be able to identify the user and actively invalidate the assigned token via the authorization token that has been sent along.

The user also has the option of deleting their account. In order to proceed with the deletion of an account, the user is required to submit a request to the designated **Delete Account** endpoint. This is achieved by sending an HTTP POST request to the `/api/auth/deleteUser`. Upon receiving the request, the server will process it and, based on the authorization token present in the request header, will proceed with the deletion of the account. Consequently, the user will no longer be able to log in with the previously defined username and password.

4.1.2. ProjectCenter API

The Projectcenter API consists of two main areas. First the initialization of the workspace data from the backend, followed by the fundamental functionalities of interacting with the workspaces, see Table 4. These include deleting, saving, and opening a workspace.

The initialization stage involves retrieving pertinent user data from the backend for display purposes. Therefore, two HTTP GET requests are sent. One for **fetching workspaces** to the path `/api/textVision/workspace/workspaces`. If the user who submitted the request is authorized, an array of all workspaces in which the applicant participates will be generated. A single workspace possesses the following attributes: an identification number, a non-unique name, a succinct description of the workspace's content and context, a list of current participants, and finally, whether the workspace is locked. The other

Table 4: Endpoints of the Projectcenter API

Functionality	Request-Line	Request	Response
fetch Workspaces	GET /api/textVision/workspace/-workspaces	{}	{id, name, description, participants, locked }
fetch Users	GET /api/textVision/workspace/user	{}	{usernames}
Open	POST /api/textVision/workspace/open	Authorization {id}	{wasLocked}
Delete	POST /api/textVision/workspace/delete	Authorization {id}	HTTP-Response-Status 200 (ok)
Save	POST /api/textVision/workspace/save	Authorization {id, name, description, participants }	{id, name, description, participants }

request is for **fetching users** at `/api/textVision/workspace/user`. If the request is authorized, an array of all usernames will be returned.

To initiate the creation of a new workspace, an HTTP POST request must be submitted to the **save** endpoint `/api/textVision/workspace/save`. This endpoint serves both the creation of new workspaces and the modification of existing ones. The request payload must include the workspace ID, name, description, and participants. If these elements are provided and the user is authorized, the server checks whether a workspace with the specified ID exists. If a matching workspace is found, two conditions are evaluated: whether the requester owns the workspace, and whether the workspace is currently in use. When either condition is true, the update is disallowed. Otherwise, the existing workspace is updated. If no workspace matches the provided ID, a new entry is created in the database. In both scenarios, if the operation completes successfully, the newly created or updated workspace is returned in the response.

To utilize a workspace, one must invoke the **open** endpoint by submitting a request to `/api/textVision/workspace/open`. The request payload contains solely the identifier of the workspace, which is used by the server to locate the corresponding record. Subsequently, the server verifies whether the requesting user is a participant in this workspace. If so, it then determines whether the workspace is currently in use, indicated by a locked state. If the workspace is not already locked, the server transitions it to a locked state, reflecting its active usage. Finally, the server responds with a boolean value indicating whether the workspace was locked before the request.

To **delete** a workspace, an HTTP request must be directed to `/api/textVision/workspace/delete`, with the workspace identifier as the sole parameter. Upon receiving this request, the server determines whether the specified ID matches an existing workspace. If so, it verifies that the requester owns the workspace and that the workspace is not locked (i.e., actively in use). If neither of these conditions applies, the workspace is deleted from the database, and an HTTP response with status code 200 is returned.

4.1.3. PDF Viewer API

The **PDF Viewer API** in TextVision facilitates the transfer, storage, and management of PDF documents between the frontend and backend. This API is responsible for handling file uploads, deletions, and annotations, ensuring efficient document interaction through a structured set of endpoints. It is important to note that the API does not provide direct viewing capabilities. Instead, the actual rendering and user interaction with PDFs is handled by the **PDFViewer** component within the frontend, which uses external libraries such as `react-pdf-viewer` to dynamically display content.

Table 5 outlines the core endpoints of the **PDF Viewer API**, detailing their functionality, request structure, and expected responses:

The **upload PDF** endpoint allows users to upload PDF files to the platform. It takes a workspace ID, file name, base64 encoded content, file size, and MIME type as input. Upon success, the API returns the newly assigned file ID along with the stored metadata. This functionality is critical in the `PDFViewer.jsx` frontend component, where users initiate file uploads and the system updates the state accordingly.

The **textttdelete PDF** endpoint is used to remove an uploaded PDF. It takes the **workspace ID** and **file ID** to identify the correct document to delete. Once processed, the backend will confirm the deletion with a success message. The frontend is then responsible for reading this response and validating it by calling its `init` function again to update its list of available PDFs accordingly.

The **save PDF note** endpoint allows users to annotate documents by adding highlights, comments, and attached images. When a user selects text and adds a note, the frontend reflects these changes in the user interface. The frontend then calls this API to persist the changes, passing the **workspace ID**, content details, and metadata to the backend. The backend saves this information and returns a unique **note ID** to confirm the successful save operation. This ensures that the user's annotations are stored securely on the server, while maintaining a responsive user experience.

The **delete PDF note** endpoint removes specific annotations. The request body includes the **workspace ID**, the **note ID**, and the associated **PDF ID**. The backend processes the request and returns a confirmation message. The frontend ensures that the deleted note

Table 5: Endpoints of the PDF Viewer API

Functionality	Request-Line	Request	Response
Upload PDF	POST /api/textVision/pdfViewer/save	{ workspaceId, fileName, content, fileSize, mimeType }	{ id, fileName, fileSize }
Delete PDF	POST /api/textVision/pdfViewer/delete	{ workspaceId, id }	{ message: Successfully deleted. }
Save PDF Note	POST /api/textVision/pdfViewer/savePDFNotes	{ workspaceId, note: { content, highlightAreas, quote, image, pdfId, marked } }	{ noteId, status: Saved successfully }
Delete PDF Note	POST /api/textVision/pdfViewer/deletePDFNote	{ workspaceId, noteId, pdfId }	{ message: Note deleted successfully. }
Mark PDF Note	POST /api/textVision/pdfViewer/markPDFNote	{ workspaceId, noteId, pdfId, marked }	{ status: Note marked successfully. }

is no longer displayed in the annotation panel.

The **mark PDF note** endpoint is used to mark notes for AI-based context recognition. Users can toggle this status, and the frontend synchronizes changes by sending a request containing the workspace ID, note ID, PDF ID, and marked status. The backend updates the note and confirms the change, allowing the frontend to effectively categorize and use marked annotations.

These endpoints provide communication between the frontend and backend components of the PDF Viewer API. They enable structured document interactions and support features such as file management, annotation, and real-time updates.

4.1.4. DocumentEditor API

The DocumentEditor API provides endpoints for managing the document editor's state; this includes saving, deleting, and retrieving the editor's content. These endpoints are essential for ensuring that users can seamlessly work on their editor state, with changes

being saved periodically and the ability to reset the editor when needed. Table 6 outlines the endpoints provided by the DocumentEditor API.

Table 6: Endpoints of the DocumentEditor API

Functionality	Request-Line	Request	Response
Save EditorState	POST /api/textVision/- documentEditor/save	{editorState, workspaceId}	{}
Delete EditorState	POST /api/textVision/doc- umentEditor/delete	{workspaceId}	{}
Retrieve EditorState	POST /api/textVision/doc- umentEditor/editor	{workspaceId}	{editorState, last- Saved}

The **Save EditorState** endpoint is used to automatically save the current state of the document editor every 30 seconds after an edit or when the user attempts to refresh the website. An HTTP POST request is sent to the path `/api/textVision/documentEditor/save`, including the current content of the editor (**editorState**) and the associated workspace (**workspaceId**). The server processes this request and saves the editor's state to the database.

The **Delete EditorState** endpoint allows users to reset the editor to its initial state, effectively clearing all content. An HTTP POST request is sent to the path `/api/textVision/-documentEditor/delete`, including the workspace (**workspaceId**). The server deletes the saved state of the editor for the specified workspace, and the frontend updates the editor to reflect this change. This endpoint can for example be used when users want to start fresh or remove outdated content.

The **Retrieve EditorState** endpoint is used to fetch the current state of the editor for a specific workspace. An HTTP POST request is sent to the path `/api/textVision/documentEditor/editor`, including the workspace (**workspaceId**). The server retrieves the saved editor state and the timestamp of the last save (**lastSaved**) and returns this data to the client. This endpoint is crucial for initializing the editor with the latest saved state when the user navigates to the editor page or refreshes the application.

4.1.5. Chat API

The Chat API is designed to facilitate primary communication through a WebSocket connection while also managing the retrieval of stored chat messages and prompt suggestions. The establishment of the WebSocket connection requires an upgrade handshake: the client initiates the process by sending an upgrade request to the URI `/api/textVision/chat/chat`, and the server responds with the appropriate handshake information. All subsequent Web-

Table 7: Endpoints of the Chat API

Functionality	Request-Line	Request	Response
WebSocket hand-shake	GET /api/textVision/chat/chat	{}	{}
Fetch Chat history	POST /api/textVision/chat/chatHistory	{workspaceId}	{[chatMessage, chatAudio, userId, chatImage, timeSent]}
Retrieve Prompt Suggestions	POST /api/textVision/chat/promptSuggestions	{prompt, modelId, temperature, workspaceId}	{[promptSuggestions]}

Socket communications are directed to this endpoint.

`Chat history` retrieval is performed via an HTTP POST request sent to the endpoint `api/textVision/chat/chatHistory`. This request includes the `workspaceId` in the header, which the backend uses to query the database. The response comprises an array of chat message objects. Each object contains several elements: the chat message as a string, an audio component (encoded in Base64 if the message was created via a microphone), a user identifier string, an image component (also Base64 encoded if images are included), and a timestamp indicating when the message was sent.

For `prompt suggestion` requests, the client must send an HTTP POST request to the endpoint `api/textVision/chat/promptSuggestions`. This request must include the following parameters: the entered prompt, a `modelId` specifying the model for generating suggestions, a temperature value between 0 and 1, and the workspace identifier. Upon validating the presence and types of these headers, the backend concatenates all PDF content from the workspace into a single text string. It then retrieves all chat messages submitted by users and computes the token count for the selected model, observing a limit of 2048 tokens. To prioritize more recent messages, the evaluation process starts with the latest message. Subsequently, a request that includes the initial prompt, the specified model, temperature settings, filtered message history, and the aggregated PDF content is sent to the AI backend. The response from the AI backend, which consists of three strings containing prompt suggestions derived from the prompt, chat history, and PDF content, is then routed back to the frontend.

4.1.6. Multimodality API

The Multimodality API in TextVision integrates speech-to-text, text-to-speech and voice command functionalities using the Web Speech API and MediaRecorder. Also, voice inputs are able to be handled via chat using the Whisper model. This section outlines the structure and functionality of the API, including both the frontend processing and backend communication.

Speech-to-Text and Text-to-Speech are designed to convert speech to text and text to speech using the Web Speech API. Through Web Speech API the user's voice is captured and then transcribed to text that is then inserted at the cursor position in the DocumentEditor. The Web Speech API also converts the text content of the DocumentEditor into speech output. Both functions are activated through the user clicking the microphone button and speaker button in the DocumentEditor. These two functions do not involve the backend, because the conversion is handled entirely on the client side using the Web Speech API.

Voice commands allows users to control the application using voice commands. Through Web Speech API captures the user's voice input and matches it against predefined commands. If a command is recognized, the corresponding action is triggered in the application. This function do not based on HTTP requests due to the command recognition and execution are handled entirely on the client side.

Chat voice messages enables users to send voice messages through the chat interface. Table 8 outlines the endpoints provided by the Multimodality API.

Table 8: Endpoints of the Multimodality API

Functionality	Request-Line	Request	Response
Voice Input Processing	Frontend Processing	transcript (speech recognition result)	new ChatMessage(transcript, null, 'user', [], null)
Voice Message Sending	POST /api/textVision/chat/chat	{workspaceId}	{[chatMessage, chatAudio, userId, chatImage, timeSent]}

Speech Input Processing: The Speech Input Processing endpoint processes speech input using the Web Speech API and MediaRecorder. The transcript of the speech recognition result is used to create a new ChatMessage object. This functionality is entirely handled on the frontend, and no HTTP request is involved.

Voice Message Sending: The Voice Message Sending endpoint allows users to send voice messages through the chat interface. An HTTP POST request is sent to the path /api/textVision/chat/chat, including the action, audio data, workspace ID, model ID,

and optional keyword. The server processes this request and returns a response containing the generated message from the AI and an error flag.

4.1.7. PromptDesigner API

The PromptDesigner API provides endpoints for managing prompts; this includes creating, saving, deleting, and evaluating prompts. These endpoints are essential for enabling users to design, optimize, and analyze prompts. The API also supports cost and token estimation of prompts. Table 9 showcases the endpoints provided by the PromptDesigner API.

Table 9: Endpoints of the PromptDesigner API

Functionality	Request-Line	Request	Response
Save Prompt	POST /api/prompt/save	{workspaceId, prompt}	{successBoolean, successMessage, savedPrompt:{...}}
Delete Prompt	POST /api/prompt/delete	{workspaceId, promptId}	{successBoolean, successMessage}
Evaluate Prompt	POST /api/prompt/evaluate	{model, framework, temperature, prompt, goal}	{issues: [...], promptRating}
Calculate Costs	POST /api/prompt/costs	{modelId, prompt}	{tokens, cost}
Solve Prompt	POST /api/prompt/solve	{model, framework, temperature, prompt, issues}	{optimizedPrompt}
Retrieve AI Models	POST /api/prompt/models	{}	{models: [...]}
Retrieve Frameworks	POST /api/prompt/frameworks	{}	{frameworks: [...]}
Retrieve Prompts	POST /api/prompt/retrieve	{workspaceId, promptId}	{prompts: [...]}

The **Save Prompt** endpoint saves a prompt to the database. An HTTP POST request is sent to the path `/api/prompt/save`, including the workspace ID (`workspaceId`) and the prompt data (`prompt`). The server processes this request and saves the prompt, returning the saved prompt object. This endpoint is used to save new prompts or update existing ones.

The **Delete Prompt** endpoint deletes a prompt from the database. An HTTP POST request is sent to the path `/api/prompt/delete`, including the workspace (`workspaceId`) and the prompt (`promptId`). The server deletes the specified prompt and returns a success response.

The **Evaluate Prompt** endpoint evaluates a prompt based on the selected AI model, framework, and temperature. An HTTP POST request is sent to the path `/api/prompt/evaluate`, including the model ID (`model`), framework ID (`framework`), temperature (`temperature`), prompt text (`prompt`), and goal (`goal`). The server evaluates the prompt and returns a list of issues and a rating. This endpoint is used to identify potential improvements in the prompt.

The **Calculate Costs** endpoint estimates the cost and token usage for executing a prompt. An HTTP POST request is sent to the path `/api/prompt/costs`, including the model (`modelId`) and the prompt text (`prompt`). The server calculates the token count and cost, returning these values in the response. This endpoint is used to provide users with cost estimates for their prompts.

The **Solve Prompt** endpoint optimizes a prompt based on the selected issues. An HTTP POST request is sent to the path `/api/prompt/solve`, including the model ID (`model`), framework ID (`framework`), temperature (`temperature`), prompt text (`prompt`), and a list of issues (`issues`). The server generates an optimized prompt and returns it in the response. This endpoint is used to apply improvements to the prompt based on the evaluation results.

The **Retrieve AI Models** endpoint fetches the available AI models that can be used for prompt evaluation and optimization. A HTTP POST request is sent to the path `/api/prompt/models`, and the server responds with a list of models. These models are used to give the user the ability to select an AI model for evaluating and optimizing prompts.

The **Retrieve Frameworks** endpoint fetches the available frameworks that define the rules for evaluating prompts. An HTTP POST request is sent to the path `/api/prompt/frameworks`, and the server responds with a list of frameworks. These frameworks are used to assess the quality of prompts and identify potential issues.

The **Retrieve Prompts** endpoint fetches the saved prompts for a specific workspace. An HTTP POST request is sent to the path `/api/prompt/retrieve`, including the workspace (`workspaceId`) and an optional prompt ID (`promptId`). The server responds with a list of prompts or a specific prompt if the prompt ID is provided.

4.2. Data-exchange between Backend and AI-Backend

The communication between the Backend and AI-Backend represents an essential interface in the TextVision architecture. Multiple endpoints enable this interaction, supporting functionalities such as prompt optimization, evaluation, chat responses, and file processing. This section presents these endpoints systematically, following the same structure used for the Frontend-Backend interface description.

It is important to note that the responses listed in the tables correspond to successful and valid requests. The Backend authenticates with the AI-Backend using Basic Authentication with username and password credentials obtained from the application's configuration.

Table 10: Endpoints for communication between Backend and AI-Backend

Functionality	Request-Line	Request	Response
Optimize Prompt	POST /optimize	{model, frame- work, temperature, prompt}	{success, optimiza- tion}
Evaluate Prompt	POST /evaluate	{model, frame- work, temperature, prompt, goal}	{success, re- sult_prompt: {issues, rating}}
Solve Prompt	POST /solve	{model, frame- work, temperature, prompt, issues}	{success, op- timization: {optimized_prompt}}
Chat Request	POST /chat_request	{message, history, vector_store_id}	{message, assis- tant_id}
File Search Request	POST /file_search_request	{message, history, vector_store_id}	{message, assis- tant_id}
Upload Files	POST /upload_files	{files: [{name, content}], vec- tor_store_id}	{vector_store_id, files: [{id, name}]}
Remove Files	POST /remove_files	{file_ids: []}	{success}
Prompt Sugges- tions	POST /promptSugges- tions	{prompt, model, temperature, ms- gHistory, pdf}	{suggestions: []}

As illustrated in table 10, the subsequent text provides an explanation of the endpoints utilized for communication between the backend and the AI-backend.

The **Optimize Prompt** endpoint allows the backend to request optimization of user-created prompts. The request is sent to the AI-Backend via an HTTP POST request to `/optimize`, containing the model name (retrieved from model ID using the `'getModelsByCustomId'` method), the framework (obtained from PDF data via `'getPDFByCustomId'`), the temperature setting, and the prompt text. The AI-Backend processes this information and returns

a success indicator along with the optimized prompt text. This endpoint is invoked when users utilize the PromptDesigner's optimization feature.

The **Evaluate Prompt** endpoint enables quality assessment of prompts against specific goals. An HTTP POST request to `/evaluate` includes the model name, framework content, temperature, prompt text, and the user-specified goal. The AI-Backend analyzes the prompt and returns a detailed evaluation containing identified issues (each with a description and improvement suggestion) and an overall quality rating. This evaluation helps users understand potential weaknesses in their prompts before practical usage.

The **Solve Prompt** endpoint applies specific fixes to prompts based on identified issues. The backend sends an HTTP POST request to `/solve` with the model name, framework content, temperature, original prompt text, and selected issues for correction. The AI backend returns an optimized prompt addressing these issues, enabling users to iteratively improve their prompts based on specific recommendations.

The **Chat Request** endpoint facilitates standard chat interactions between users and the AI system. The backend sends an HTTP POST request to `/chat_request` containing the user's message, the conversation history (formatted as an array of role-content pairs), and an optional vector store ID if relevant. The AI backend processes this request and returns a response message along with an Assistant ID that can be used for conversation continuity. This endpoint supports the core chat functionality within the application.

The **File Search Request** endpoint enables context-aware conversations based on uploaded documents. The backend sends an HTTP POST request to `/file_search_request` with the user's message, conversation history, and a vector store ID that references previously uploaded documents. The AI-Backend utilizes this context to generate responses that incorporate information from the relevant files. This endpoint is crucial for the application's document-grounded conversation capabilities.

The **Upload Files** endpoint allows the backend to transmit PDF files to the AI-Backend for vectorization and storage. An HTTP POST request to `/upload_files` contains an array of files (each with a name and base64-encoded content) and an optional vector store ID for appending to existing collections. The AI-Backend processes these files and returns a vector store ID along with an array of processed files, each with its own ID and name. This vectorization enables the AI to reference and use document content in conversations.

The **Remove Files** endpoint enables deletion of files from the AI-Backend's storage. The backend sends an HTTP POST request to `/remove_files` with an array of file IDs to be removed. The AI-Backend processes this request and returns a success indicator. This endpoint is used when users delete PDFs from the workspace to ensure synchronization between the backend database and the AI-Backend's vector store.

The **Prompt Suggestions** endpoint provides context-aware prompt suggestions based on workspace history. The backend sends an HTTP POST request to `/promptSuggestions` containing the current prompt draft, model name, temperature setting, message history from the workspace, and aggregated PDF content if available. The AI-Backend generates relevant suggestions tailored to the workspace context. This endpoint enhances the user experience by offering intelligent completion options when designing prompts.

5. Frontend

This chapter explores the various aspects of frontend development, beginning with an overview of folder structure, UI layout, routing, and communication with the backend. It then delves into key features such as authentication, document editing, multimodality, and memoization for optimized rendering. It also provides insight into error handling, user feedback mechanisms, and testing strategies to ensure robustness.

By understanding the concepts outlined in this chapter, readers will gain a comprehensive view of how the frontend is designed and implemented, paving the way for further exploration of specific component libraries and advanced features discussed in later sections.

5.1. Overview of the Frontend

This section provides an introductory overview of the TextVision Frontend. It begins by illustrating the hierarchical organization of the Frontend directories and stating the rationale underlying this structural choice. Subsequently, the user interface layout is described, including the justification for its specific implementation. The discussion then advances to the routing mechanism, highlighting the employment of React-Context to manage access control. Finally, the section introduces the Axios and WebSocket libraries, which serve as the communication bridges to the Backend.

5.1.1. Folder Structure

The folder structure exemplifies a modular and scalable client-side application design. At the top level, the project ("pg-client") is divided into core source files and test suites. Within the "src" directory, the codebase is systematically organized into specialized sub-directories:

```
pg-client/  
|-- src/  
    |-- api/  
    |-- assets/  
    |-- components/  
    |-- context/  
    |-- hooks/  
    |-- models/  
    |-- pages/  
|-- tests
```

- **api/**: Contains modules responsible for handling network requests and interfacing with external APIs.
- **assets/**: Houses static files such as images, fonts, and stylesheets.

- **components**/: Encapsulates reusable UI elements, adhering to a component-based architecture.
- **context**/: Manages application-wide state through context providers, facilitating state sharing without prop drilling.⁶
- **hooks**/: Contains custom hooks that encapsulate and abstract reusable logic, enhancing code modularity.⁷
- **models**/: Defines data structures and types, ensuring consistency in how data is represented throughout the application.
- **pages**/: Organizes view components corresponding to distinct application routes or pages.

Finally, the "tests" directory, located at the root level, is dedicated to automated testing, ensuring code quality and robustness through unit and integration tests. This structured approach promotes maintainability, reusability, and clear separation of concerns across the project.

5.1.2. UI Layout

TextVision's user interface is designed with a structured and modular approach using Bootstrap⁸. Bootstrap's predefined styles and components ensure a consistent design across all pages, making it easier to maintain a consistent look and feel throughout the application. This design framework allows for effortless customization and scalability. In the context of TextVision, this allows users to work in a single integrated workspace that combines the PDF viewer, document editor, and chat interface. By eliminating the need to switch between multiple tabs or applications, users can focus on content creation and benefit from AI-powered document refinement. The system uses a React application with context providers and custom hooks to manage state and enable data exchange between components. It also uses Bootstrap to ensure consistency, responsiveness, and adaptability across devices.

The homepage serves as the primary workspace, where users interact with documents, collaborate, and generate new content. Figure 26 shows that the layout is divided into three main sections.:

The Sidebar Navigation, positioned on the left, provides access to key sections such as Home, PromptDesigner, FAQ, Profile, and Debug. It is designed to be collapsible to maximize screen space when needed. Using Bootstrap's styling classes, the sidebar is structured for intuitive navigation while maintaining a clean design. The main content area uses Bootstrap's grid system with the `row` and `col` classes to create a flexible and proportional layout. This section consists of:

⁶<https://react.dev/learn/passing-data-deeply-with-context>

⁷<https://react.dev/learn/reusing-logic-with-custom-hooks>

⁸Contributors, M. O. J. T. a. B. (n.d.). Bootstrap. <https://getbootstrap.com/>

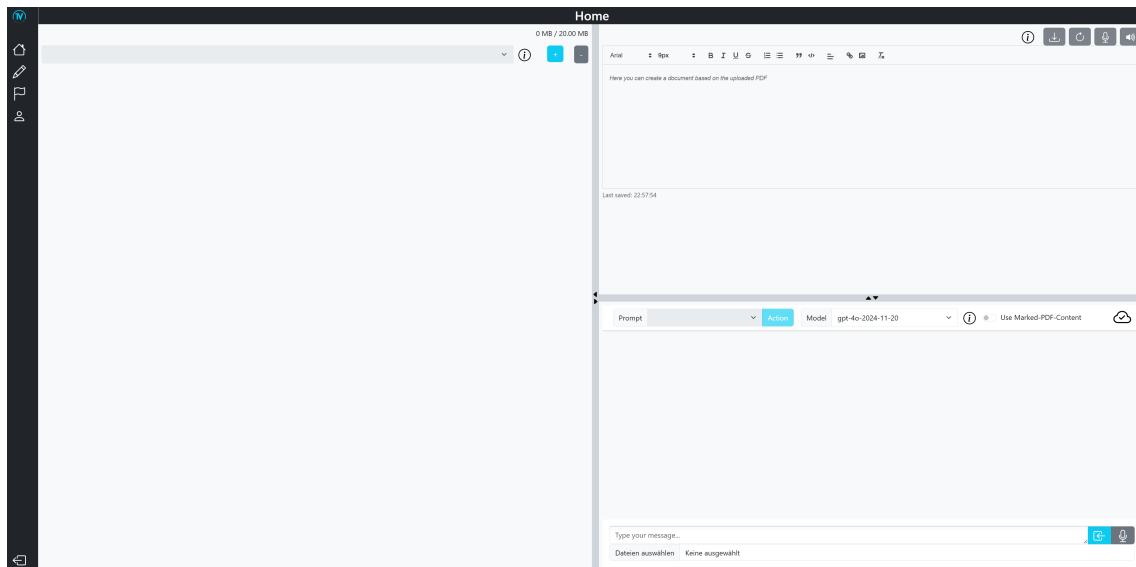


Figure 26: Home-Page of TextVision

- **PDF Viewer**, which supports PDF uploads, page navigation, text selection, and AI-assisted annotations. The viewer, powered by *react-pdf-viewer*, enables text extraction, highlighting, and efficient document interaction.
- **Document Editor and Chat Interface**, structured using Bootstrap's grid system. The WYSIWYG Document Editor allows real-time editing and AI-assisted content generation, while the Chat Interface enables AI-driven discussions, with responses that can be directly inserted into the editor.

The Header Bar, built with Bootstrap's `navbar`, spans the top of the interface and provides global navigation, workspace selection, and quick access to user settings. It includes Bootstrap Icons (`bi`) for enhanced visual guidance.

Bootstrap plays a crucial role in the design consistency of TextVision. The `container-fluid` class ensures full-width customization, while the `row` and `col` classes maintain proportional spacing between UI components. Standardized buttons, modals, and form controls provide a consistent look and feel across different areas of the platform.

Integrating Bootstrap provides several benefits:

- Consistent UI elements, ensuring uniformity across all buttons, forms, and tooltips.
- Dynamic Theming, allowing color and layout customization using Bootstrap's SASS variables.
- Responsive design, allowing the site to seamlessly adapt to different screen sizes with minimal adjustments.

- Accessibility enhancements, such as built-in ARIA attributes and keyboard navigation support.

TextVision’s modular architecture is optimized for performance and maintainability. The system uses lazy loading for non-essential components and asynchronous API requests to prevent UI lag. In addition, WebSocket communication ensures real-time updates in the chat interface, improving response times and fluidity of user interaction. The Document Editor features an autosave mechanism to prevent data loss, while the chat system caches AI responses to improve retrieval efficiency.

Each component operates independently, but is connected through React’s context providers, minimizing unnecessary re-rendering and improving overall application performance.

5.1.3. Routing and Context

Figure 27 shows the component-based routing system of TextVision, in which each URL endpoint is explicitly mapped to a distinct page component, thereby enabling a clear and modular navigation structure. Routing is declared through a hierarchical configuration that supports nested paths, ensuring that related content is logically grouped. Complementing the routing mechanism, context providers such as the `AuthenticationContext` and `ProjectContext` manage state-specific concerns. The `AuthenticationContext` governs user authentication, session management, and access restrictions, while the `ProjectContext` encapsulates project-related data and enforces access control by verifying the presence of a valid workspace in persistent storage before permitting navigation to project-specific routes. This integrated approach of combining routing with dedicated context providers promotes a robust, scalable, and maintainable application architecture.

Route and Pages

The routing layer is designed to map specific URL paths to corresponding page components. Each route defines a unique view, ensuring that the application can render contextually appropriate content based on the user’s navigation. Key points include:

- **Hierarchical Structure:** Routes such as `/faq` include nested paths (e.g., `/faq/profile`, `/faq/first-steps`), which provide a structured navigation experience that logically groups related content.
- **Component-Based Routing:** Each route is associated with a particular component or page, such as the Home page, PDF-Viewer, Chat Editor, and specialized sections like PromptDesigner. This modular approach allows for isolated development, testing, and maintenance of individual components.

AuthenticationContext

The `AuthenticationContext` is a dedicated state management layer that centralizes all aspects of user authentication (see Section 5.2). Its primary functions include:

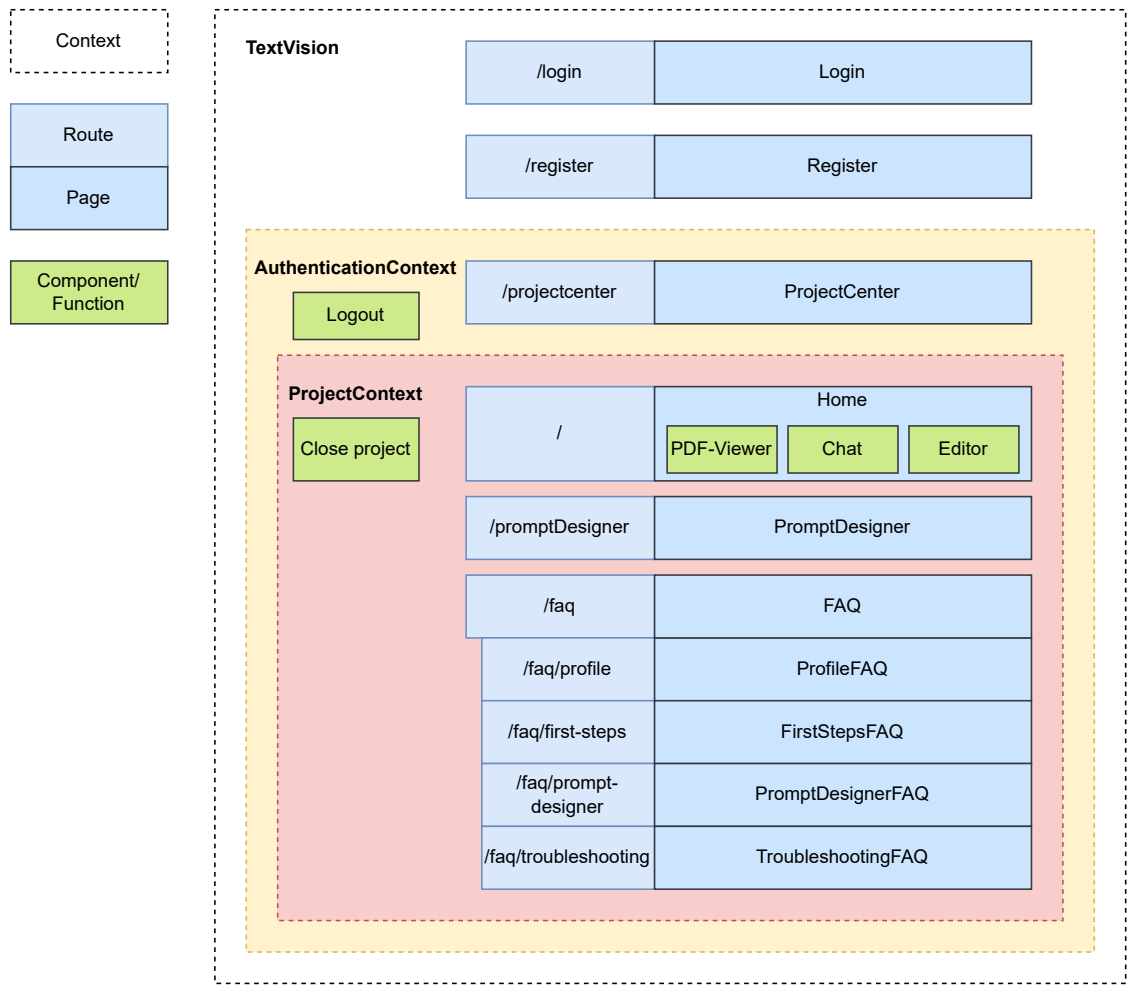


Figure 27: Frontend: Structure and Routing

- **User Verification:** It handles the processes involved in verifying user credentials during login and registration, ensuring that only authorized users gain access to protected routes.
- **Session Management:** By maintaining state information such as tokens or session identifiers, it facilitates persistent login sessions and supports seamless navigation across different parts of the application.
- **Access Control:** It enforces access restrictions on certain routes (e.g., project center or profile-specific pages), ensuring that sensitive or personalized content is only available to authenticated users.

ProjectContext

The `ProjectContext` serves as a dedicated management layer for project-specific state within the application. Its responsibilities include:

- **State Isolation:** It encapsulates all data and operations related to projects, such as project details, active project selection, and modifications, ensuring that project data is managed separately from other application concerns.
- **Data Consistency:** By centralizing project-related data, the `ProjectContext` ensures consistency across different components that require access to the same project information, reducing redundancy and potential data conflicts.
- **Access Control:** A crucial feature of the `ProjectContext` is its built-in access control mechanism. This check verifies if a workspace is stored in the persistent storage before allowing routing to project-specific pages. If the workspace is absent, the context prevents navigation to these pages, thereby safeguarding the application from accessing project functionalities without the required environment. This ensures that users only access project routes when a valid workspace is present, maintaining the integrity and context-specific flow of the application.
- **Contextual Integration:** This context integrates seamlessly with the routing system, ensuring that when users navigate to project-specific pages (e.g., the Project-Center), the necessary project data is readily available and managed in a uniform manner.

5.1.4. Communication to Backend

As elucidated in Chapter 3, communication with the backend occurs via both HTTP and WebSockets. The Axios library (`axios-http`) is employed for communication with the HTTP protocol, while JavaScript's integrated functions are utilized for WebSocket communication. Both of these methods are outlined in the following sections.

Axios (`axios-http`)

Axios (`axios-http`) is a JavaScript-based HTTP client library that is primarily used in web development for communicating with APIs. It is characterized by a simple syntax, promise-based processing, and support for asynchronous operations. The main functions include support for HTTP methods (see Section 2.3.2), automatic processing of JSON data, and the ability to manipulate requests and responses globally using interceptors. Axios facilitates the configuration of headers, the imposition of time limits, and the cancellation of requests, a feature that is particularly advantageous in data-intensive or distributed systems. Configurations can be defined on an instance basis.⁹

⁹<https://axios-http.com/docs/intro>

WebSocket

The WebSocket API is a critical component in modern web development, enabling efficient, bidirectional communication between clients and servers over a single, persistent TCP connection. By establishing a dedicated communication channel, the WebSocket protocol minimizes overhead compared to traditional HTTP requests, thereby reducing latency and resource consumption. This API supports real-time data transfer, allowing applications to send and receive both textual and binary data, which is particularly advantageous in scenarios such as live chat, collaborative editing, online gaming, and financial trading. The underlying protocol begins with an HTTP-based handshake that upgrades the connection to a WebSocket, ensuring a smooth transition to a full-duplex communication model. Its event-driven architecture, which includes event handlers for connection open, message receipt, errors, and closure, further enhances its adaptability and responsiveness in dynamic application environments.¹⁰

5.2. Authentication

This section provides a comprehensive overview of the user authentication process, encompassing the creation and deletion of user accounts, as well as the procedure for logging out. In essence, the endpoints to the backend, as delineated in Section 4.1.1, are elucidated in conjunction with the user interface. Given that all endpoints are HTTP requests, the HTTP client Axios (axios-http), as outlined in Section 5.1.4, is employed for communication with the backend.

In addition to the functions described, selected special features of the frontend implementation are described below, including the automatic configuration of the JWT received in the authentication process (see Section 2.3.6 Authentication process) for subsequent HTTP requests using Axios, and considerations for persistent login.

5.2.1. Register

To utilize TextVision, the user must possess a user account, which can be established using the register form. This form comprises two input fields, username, and password, in addition to a checkbox that determines whether the data should be saved or not (for further details, refer to the persistent login section in Section 5.2.5). Furthermore, a button is available for confirming or sending the data. The password input field is accompanied by a button that determines whether the password should be masked or displayed in plain text. Additionally, a link is present below the form that leads to the login function (see Section 5.2.2). The complete form can be viewed in Figure 28.

As demonstrated in the accompanying illustration, the red text situated beneath the user name input field serves to validate the input fields on the front end. This validation is

¹⁰https://developer.mozilla.org/de/docs/Web/API/WebSockets_API

Register

Username

Must be between 6 and 16
characters long.

Password



☒ Remember me

Register

[Already registered?](#)

Figure 28: User Interface: Register

achieved through the utilization of regular expressions (regex). The following regular expressions are employed:

```
1 {  
2 const REGEX_USERNAME = /^[0-9A-Za-z]{6,16}$/;  
3 const REGEX_PASSWORD = /^(?=.*?[0-9])(?=.*?[A-Za-z]).{8,32}$/;  
4 }
```

Sourcecode 4: Register Regex-Validation

The user name must comprise between 6 and 16 characters, whilst the password must consist of between 8 and 32 characters, with each character type being present once. In the event that the regular expression conditions are not met, the confirmation button (illustrated in blue with the text 'Register') is deactivated.

Actuation of the button instigates the transmission of an HTTP POST request to the Backend, as delineated in Section 4.1.1 of the description Register. In the event of a successful request, the tokens contained within the response are saved, thereby ensuring the successful creation of a user account, successful login, and simultaneous authentication of the user.

The storage and utilization of the tokens received during the authentication process are elucidated in greater detail in Section 5.2.5.

5.2.2. Login

The authentication process via login bears a strong resemblance to the register function (see Section 5.2.1), with the forms displaying a high degree of similarity, the only notable differences being the presence of customized texts and the absence of regular expression validation. The link situated beneath the form directs the user to the register function upon logging in and is accompanied by the caption 'No Account?'. Following the completion of a successful HTTP POST request to the designated backend path (`api/auth/login`), as outlined in Section 4.1.1, the authentication tokens are stored and utilized in a manner consistent with the register process.

5.2.3. Logout

It is important to note that authenticated and thus logged-in users have access to the other functionalities of TextVision, as described in Section 5.1.3 and the following Section 5.2.5. The user name of the account with which the user has logged in can be seen in the sidebar (see Section 5.1.2) under the profile icon. This also includes the Logout and Delete Account (see Section 5.2.4) functions. Both functions are displayed as buttons. Figure 29 illustrates this using an excerpt.

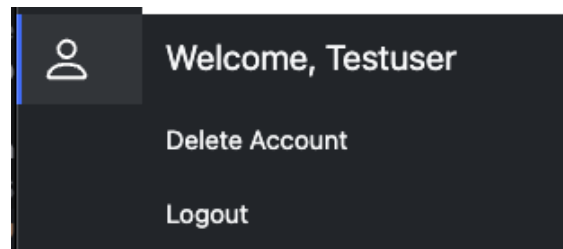


Figure 29: User Interface: User Info

Should the user elect to log out, an HTTP POST request is initiated towards the backend. According to the descriptions provided in Section 4.1.1, this results in the backend discarding the user's authentication tokens. Concurrently, the user's authentication tokens are also deleted in the front end, and the user is redirected to the login page (see Section 5.2.2). Consequently, the user is no longer able to access pages other than register and log in without undergoing a new authentication procedure (see Section 5.1.3 and Section 5.2.5).

5.2.4. Delete Account

The function designated as 'Delete Account' has been developed to remove the specified user account. As illustrated in Figure 29, the option is located in the sidebar. Following

the selection of the button, the intended behavior is confirmed using the dialogue shown in Figure 30.

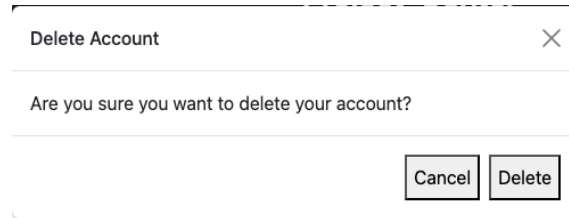


Figure 30: User Interface: Confirmation Delete Account

Upon confirmation of the dialogue, a request is initiated towards the backend, consequently leading to the deletion of the user account. In a manner analogous to the logout procedure, the user is redirected to the login page and consequently lacks access to the TextVision functionalities without the necessity of re-authentication (see Section 5.1.3 and Section 5.2.5).

5.2.5. Technical specialties

In the preceding sections, the discussion centered on the individual functionalities and their linkage to communication with the backend (see Section 4.1.1) and the general theoretical authentication process (see Section 2.3.6). The present section will undertake a more detailed examination of the technical features.

These include the storage of the authentication status, access restrictions and persistent login, as well as the execution of further authenticated requests.

Storage of authentication-state

Upon completion of the user registration process, which can be initiated through the login or register functions, the authentication tokens (JWT) are stored. The React-Context functionality facilitates information storage, enabling its accessibility to specific components in interaction with a provider. These components possess the capability to both read and write the information.¹¹

Due to the importance of authentication, this context encompasses all application components. The login, registration, and logout functions are all contingent on successful authentication. After a successful logout, the information is written or deleted. The remaining components are solely responsible for reading information. This is of paramount importance for both the transmission of subsequent authenticated requests to the backend and the management of access rights within the application. These points will be elaborated upon in greater detail subsequently in this section.

¹¹<https://react.dev/reference/react/createContext>

In addition to the context functionality, the authentication tokens are stored in the local memory of the browser used. This means that the information is also available or available again when the application is closed (independent of runtime). Figure 31 illustrates this using the development tools of the Chrome browser¹².

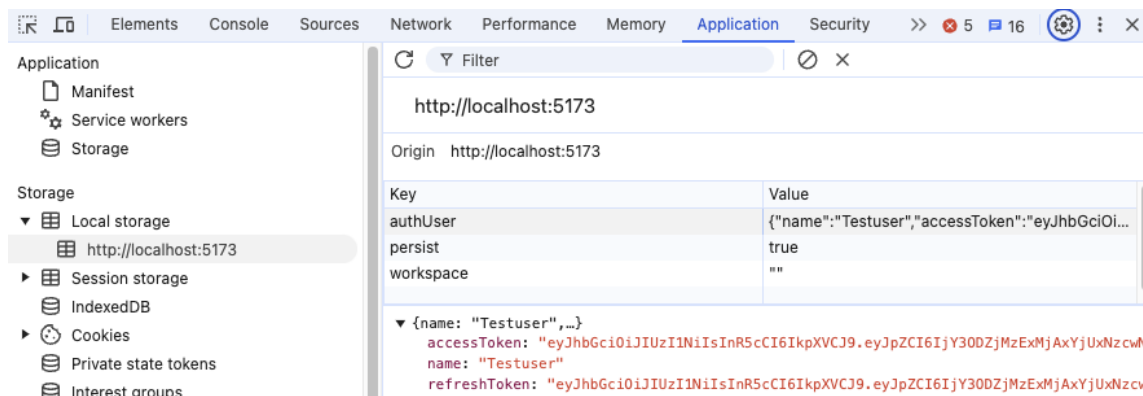


Figure 31: Browser local storage: authentication, persist and workspace

As illustrated, three pieces of information are stored in the local memory: the authentication data (**authUser**), the selection of whether the user has activated persistent login (**persist**), and the current project in which the user is located, if available (**workspace**). It is also evident that the storage is application-specific, as the screen was recorded during development. TextVision has the URL **http://localhost:5137**.

The runtime-independent storage described herein is of crucial importance to achieve persistent login. To facilitate persistent login, the browser's local memory is used to store both the authentication data and the persist field. Further details about this process can be found in the following section.

RequireAuth - Access rights through authentication

Authentication plays a pivotal role in determining access to the application, as outlined in Section 5.1.3. This section previously established that a user's access to most of the application is contingent upon successful authentication. Figure 27 in Section 5.1.3 provides a visual representation of this concept by showcasing the `AuthenticationContext`, encompassing the `ProjectCenter` and the project-specific pages (`Home`, `PromptDesigner`, and `FAQ`). It should be noted that the `AuthenticationContext` corresponds to the context mentioned in the previous section concerning saving the authentication data, except that setting the data via `login` and `register` is not included.

Access control to the area in the Authentication Context is facilitated by a component known as RequireAuth. This component encompasses the routing to the access-protected pages and verifies the user's authentication upon accessing a route.

¹²<https://developer.chrome.com/docs/devtools>

```
1 export default function RequireAuth() {  
2   const { loggedIn, auth } = useAuth();  
3   const location = useLocation();  
4  
5   return loggedIn && auth ? (  
6     <Outlet />  
7     ) : (  
8     <Navigate to="/login" state={{ from: location }}  
9       replace></Navigate>  
10  );  
}
```

Sourcecode 5: RequireAuth

In Sourcecode 5, the stored authentication data and the information as to whether the user is logged in is read by the `useAuth` function (line 2). If the user is logged in and authenticated, the condition in line 5 is fulfilled and the desired route is called. The route is displayed using the outlet element (`<Outlet />`), which is specific to React (react-router-dom)¹³. If the condition is not met, the user is automatically redirected to the `/login` route and thus to the login page. Access to the desired route is prevented and authentication/login is requested.

Persistent login and refresh token

Upon logging in and registering, the user has the option of activating a persistent login by selecting a checkbox. If this action is undertaken during the authentication process, the pertinent information is stored in conjunction with the authentication data in the browser's local storage. This runtime-independent storage is imperative for restoring authentication if the browser tab of the application is closed and reopened, and/or another application has been accessed in the same tab in the meantime.

As soon as the application is started, a check is conducted of the local storage to determine whether persistent login has been configured. If it has been configured, the authentication data is then validated using the endpoint `/api/auth/refresh-token`, as outlined in Section 4.1.1. This is a prerequisite due to the finite nature of the JWT, which is subject to expiration after a designated period. The transmission of a potentially expired JWT alongside the refresh token enables the Backend to validate it, thereby facilitating the receipt of updated authentication data in response, which is valid and can be utilized for subsequent requests, should the update be successful. Conversely, in the event of a failed update, the user is redirected to the login page in a manner analogous to authentication-based access control, necessitating re-authentication.

¹³<https://reactrouter.com/home>

Further requests with authorization

As was emphasized in the preceding section, and as was also explained in Section 2.3.2 in the context of HTTP and Section 2.3.6 in the context of the general authentication process, authentication is important for further requests to the Backend. The authentication token (JWT) is sent as part of the HTTP header with every request and enables the backend not only to recognize the sending user but also to validate whether the user has access to the requested function (see Section 2.3.6).

In the context of authenticated requests to the backend, the implementation employs the option of multiple Axios instances. Each instance of Axios is capable of configuring in a manner that is distinct from its peers, with the result that the configuration has a discernible effect on the HTTP requests that are sent.¹⁴ The following two Axios instances have been defined and utilized for HTTP communication with the backend:

The initial Axios instance, designated `axios`, is responsible for requests that do not necessitate authentication. These include the register and login functions. It is important to note that no authentication data is available for either of these requests, as the user only receives this as part of the request.

The second Axios instance, designated `axiosPrivate`, is reserved for all requests necessitating authentication. This instance applies to all pages and components within the AuthenticationContext delineated in Figure 5.1.3.

The authentication data for `axiosPrivate` must be configured within the HTTP header for each request. This process is automated through the implementation of HTTP interceptors for `axiosPrivate`, which are capable of modifying both outgoing requests and incoming responses. Figure 32 offers a visual representation of this concept, depicting an interceptor as a box during the communication process. In essence, communication occurs through the defined interceptor, which can both read and modify requests and responses.

In the context of `axiosPrivate`, an interceptor for outgoing requests has been defined in such a manner that it performs a check to ascertain whether the authentication data has been set in the header. If authentication data has not yet been set, it is the function of the interceptor to set the authentication data to active. The consequence of implementing this is that it prevents the sending of requests without authentication.

The function of the interceptor is furthermore to intervene in the communication process of incoming responses if the response has an HTTP status code of 403 (see Section 2.3.2). This status code indicates that access to the requested resource has been denied, and in such cases, the interceptor updates the authentication data using the refresh-token endpoint (see Section 4.1.1) and resends the originally sent request for which the response was intercepted with the updated authentication data. It is also noted that the interception and resending of the request are saved to avoid repeated interception of the response in question. The interceptor for concurrent responses primarily handles the case when the authentication data has expired; in this instance, the first request with outdated authen-

¹⁴<https://axios-http.com/docs/instance>

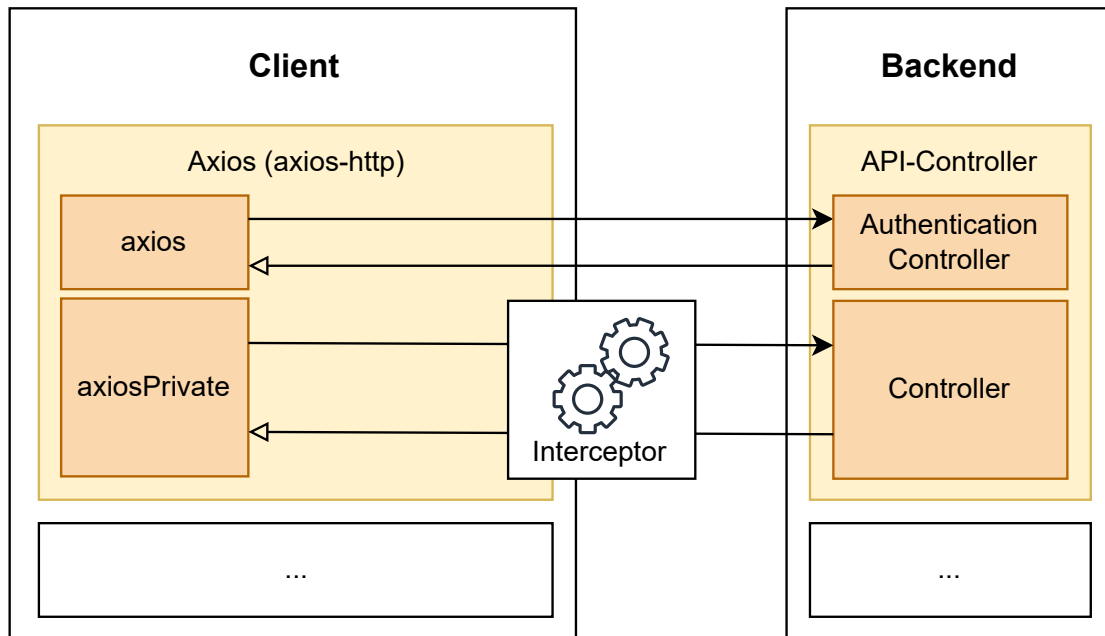


Figure 32: HTTP-Interceptor: AxiosPrivate-Instance communication with the Backend

tication data is automatically updated and the request is made despite being rejected for the first time.

5.3. ProjectCenter

This section introduces the ProjectCenter, which constitutes the initial interface presented to users following successful registration or login. Its primary function is the management of workspaces, encompassing activities such as creation, updating, deletion, and opening of workspaces. The section is organized in two parts: the first part describes the internal representation of workspaces, and the second part delineates the functional capabilities of the Projectcenter.

Workspaces

Within application contexts, workspaces are defined as virtual environments that facilitate the systematic organization of tools, projects, and configurations to optimize user workflows. In the Textvision framework, each workspace is composed of a PDF Viewer, Chat interface, PromptDesigner, and Document Editor. Modifications made to any of these individual components are uniquely bound to the respective workspace and are persistently stored, thereby enabling users to transition between multiple workspaces while preserving the distinct state of each.

Internal Representation of Workspaces

In the frontend implementation, a workspace is represented by the `Workspace` class, as detailed in Table 11. This class encapsulates the following attributes:

- **id**: unique identifier of the workspace.
- **name**: non-unique name of the workspace.
- **description**: description of the workspace.
- **participants**: A list of participants of the workspace.
- **locked**: indicates whether the workspace is in use and therefore locked.

These attributes correspond directly to the data provided by the server in response to the HTTP request for fetching workspaces (see Section 4.1.2).

The class defines two principal functions:

- **createTemplate**: This function generates an initial workspace template. It instantiates a workspace with an id of -1, a default name "New Project", an empty description, and assigns the provided `authUser` as the owner. The use of an id of -1 is deliberate; because the workspaces are sorted in ascending order based on the id, the template is always positioned at the beginning of the list. This design also ensures that a template is displayed even when no saved workspaces exist.
- **loadFromData**: This function processes an array of objects obtained from the server's HTTP response from fetching workspaces. Each object in the array contains the attributes id, name, description, participants, and locked. The function iterates over the array, creating a new instance of the `Workspace` class for each object.

Table 11: Simple UML-like class diagram for the `Workspace` class

Workspace	
+ id : number	
+ name : string	
+ description : string	
+ participants : Participant[]	
+ locked : boolean	
+ createTemplate (<i>authUser: Object</i>) : Workspace	
+ loadFromData (<i>dataArray: Object[]</i>) : Workspace[]	

Structure and functionality of the ProjectCenter

Figure 33 illustrates the ProjectCenter, which is divided into two distinct sections. The first section, termed the "Select Project" area, comprises a drop-down menu listing the names of the retrieved workspaces. Selecting a workspace name triggers a transition in

the workspace template to correspond with the chosen workspace. Adjacent to the drop-down menu, an "Open" button is provided. When activated, this button initiates an HTTP POST request to `api/textVision/workspace/open` (see Section 4.1.2). This request verifies the lock status of the workspace, and if the workspace is not locked, the user is subsequently redirected to the Homepage.

The second section, referred to as the "Edit Project" area, permits the user to modify the name, description, and participant list of the selected workspace. Any alterations can be preserved by activating the green "Save" button, located at the bottom of the interface. This action results in an HTTP POST request being sent to `api/textVision/workspace/save` (see Section 4.1.2), thereby committing the changes to the backend. Furthermore, if the selected workspace contains saved modifications, a red "Delete" button is available to facilitate workspace deletion. Activation of the delete button dispatches an HTTP POST request to `api/textVision/workspace/delete` (see Section 4.1.2). Upon successful deletion, the interface reverts to the default workspace template.

The screenshot displays the 'Project-Center' interface. At the top, there's a 'Select Project' section with a dropdown menu showing 'Project' and 'First', and an 'Open' button. Below this is the 'Edit Project' section. It has a 'Name' field with 'First' and an 'i' icon. The 'Description' field contains 'This is the first Project' with an 'i' icon. The 'Participants' section has a header 'Participants:' and an 'i' icon. Below it, there's a text prompt 'Select the user that are allowed to work on this project.' and two buttons: 'Check all' and 'Uncheck all'. A section titled 'Amount of participants: 1' follows, with the text 'Besides you, the following users participate in the project:'. Below this is a list of users with checkboxes: 'testUser1', 'testUser2', 'testUser3', 'LeonGoogle' (which is checked and highlighted in blue), and 'LeonSelzer'. At the bottom of the 'Edit Project' section are three buttons: 'Save' (green), 'Delete' (red), and 'Reset changes' (blue).

Figure 33: GUI of the ProjectCenter

5.4. PDF Viewer

The PDF Viewer in TextVision is a comprehensive tool that leverages the react-pdf library to integrate PDF document viewing and interaction into the application, as shown in Figure 34. This component is built using the **react-pdf**¹⁵ library. This component is built using the react-pdf-viewer library, which was chosen for its broad feature set,

¹⁵React-pdf. (n.d.). Retrieved February 12, 2025, from <https://react-pdf.org/>

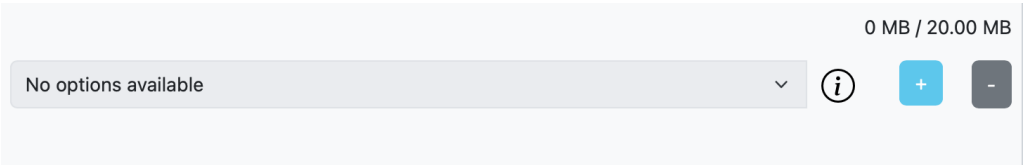


Figure 34: PDF Viewer component

customization options, and open source status¹⁶. It serves as a solid foundation for document analysis, annotation, and AI-driven interactions, while allowing for integration with React applications. The library’s plugin architecture allows for the extension of functionality, which is beneficial when implementing features such as highlighting, annotation, and AI-assisted content analysis.

In particular, the PDF Viewer is designed to facilitate comprehensive document analysis by providing users with a range of capabilities. Users can upload and render PDF files directly from within the application, using react-pdf’s viewer component for smooth rendering. The PDF Viewer is designed to facilitate document review and analysis by allowing users to load PDFs, navigate through multi-page documents, and adjust zoom levels, as shown in Figure 35.

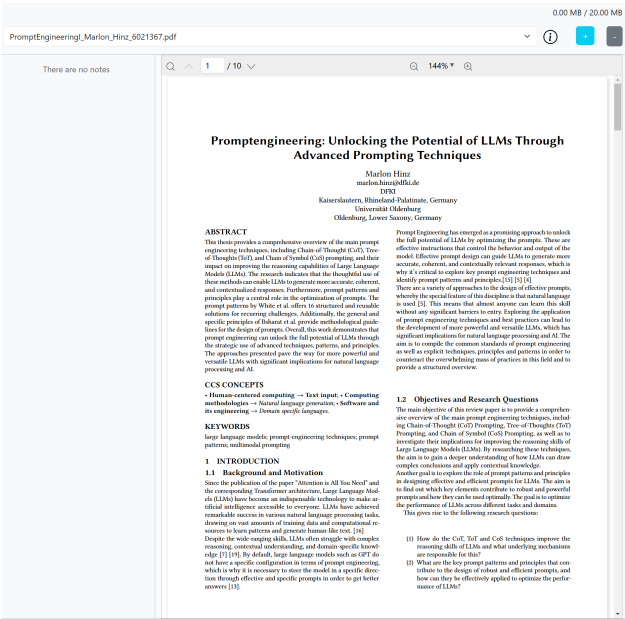


Figure 35: PDF Viewer after document upload

¹⁶Masharipov, H. (2025, January 30). Comparing the best React PDF viewers for developers — Nutrient. Nutrient. <https://www.nutrient.io/blog/top-react-pdf-viewers/>

One of the key features is the ability to highlight and annotate text, implemented using the selection and event handling capabilities of the react-pdf-viewer library. Users can select specific passages within the document to create contextual notes that are dynamically stored and retrieved through a custom backend implementation. This feature is particularly useful for in-depth document analysis and review.

The PDF Viewer uses highlighted text as context for AI-powered chat functionality, allowing users to leverage selected content for collaborative AI-driven discussions. This integration, which builds on the PDF Viewer’s text selection capabilities, enables deeper document analysis and provides users with intelligent insights based on the selected content. The react-pdf-viewer library provides the foundation for text selection, while the AI integration works as a separate function that uses the selected text as input.

A simplified overview of the PDF Viewer component, including its attributes, methods, and dependencies, is shown in Figure 36. This diagram visually represents how the component integrates various libraries, manages state, and interacts with external APIs.

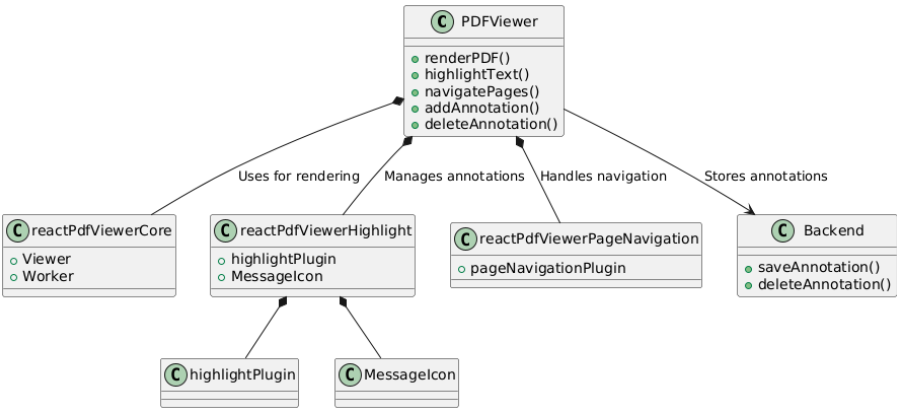


Figure 36: Simplified diagram of the PDF Viewer Component

The PDF Viewer component relies on `reactPdfViewerCore` for rendering, while annotation capabilities are provided by `reactPdfViewerHighlight`. Within this package, the `highlightPlugin` handles text selection and highlighting, while the `MessageIcon` ensures visual feedback for annotations. Navigation is facilitated through the `reactPdfViewerPageNavigation` module, which enables seamless movement between pages. For data persistence, annotations are stored via the `Backend`, which processes note creation and deletion requests. This ensures that user-generated content remains accessible.

The implementation of the PDF Viewer incorporates several technical features to ensure optimal performance and functionality. The `pageNavigationPlugin` enhances document navigation, allowing users to move between pages efficiently. The `highlightPlugin` powers the annotation system, enabling text selection and tooltip-based note creation. To

effectively manage the state of uploaded files and annotations, the component integrates a custom PdfContentContextfor structured data handling.

In terms of security, the PDF Viewer manages file uploads and deletions through authenticated API requests using axiosPrivate, preventing unauthorized access and maintaining user data integrity. Moreover, annotations are stored and retrieved through structured data handling, ensuring that users can interact with their documents efficiently. Users can create notes by selecting text, after which tooltips provide an interface for annotation input. Figure 37 outlines the process for handling annotations, including note creation, retrieval, and deletion. The diagram below outlines the interactions involved in this process.

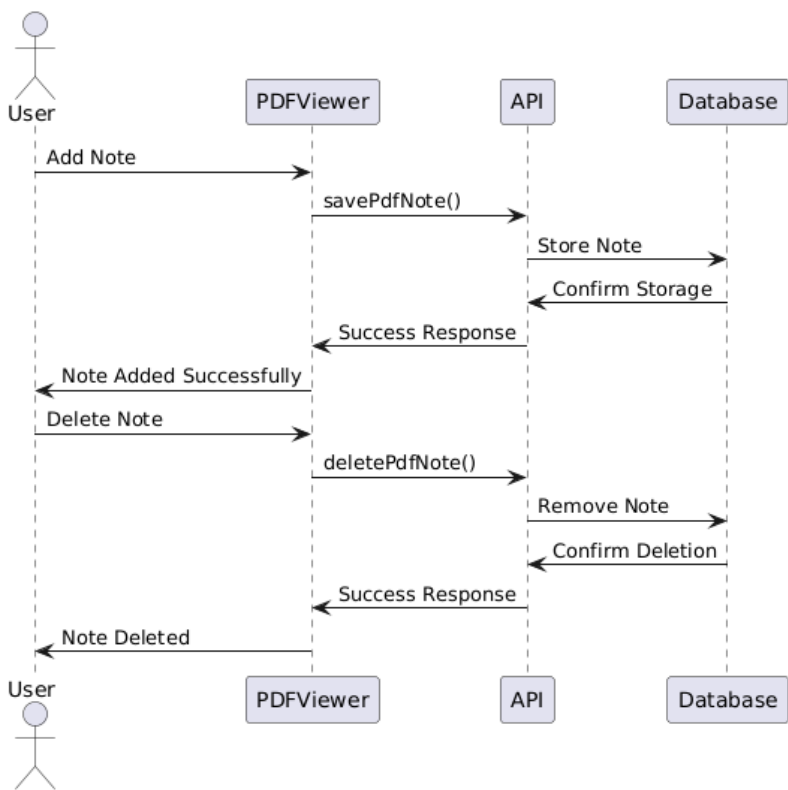


Figure 37: Diagram for PDF Annotation Handling

To streamline the annotation process, the highlightPlugin API implements note creation tooltips that allow the users to add annotations as they hover over selected text. In addition, built-in chat functionality allows users to send selected text to an AI assistant for contextual analysis and response generation.

Recognizing the importance of user control, the PDF Viewer Component includes the ability to abort ongoing annotations or chat interactions before submission, giving users the flexibility to refine their actions. For more precise control, users can selectively remove previously highlighted and tagged notes to maintain a clean, relevant set of insights.

The PDF Viewer also includes responsive UI scaling, which dynamically adjusts the layout based on the viewport size, ensuring an optimal viewing experience across screen sizes.

5.5. DocumentEditor

This section provides a comprehensive overview of the `DocumentEditor`'s functionality, which allows users to create, edit, and manage documents within TextVision. The `DocumentEditor` (see Figure 47) is a rich text editor that supports various formatting options, text-to-speech, speech-to-text, and the ability to export documents as PDFs. The editor's state is persistently saved and synchronized with the backend, ensuring that users can always resume their work. The communication with the backend is handled via HTTP requests using Axios. The `DocumentEditor`'s endpoints are described in Section 4.1.4.

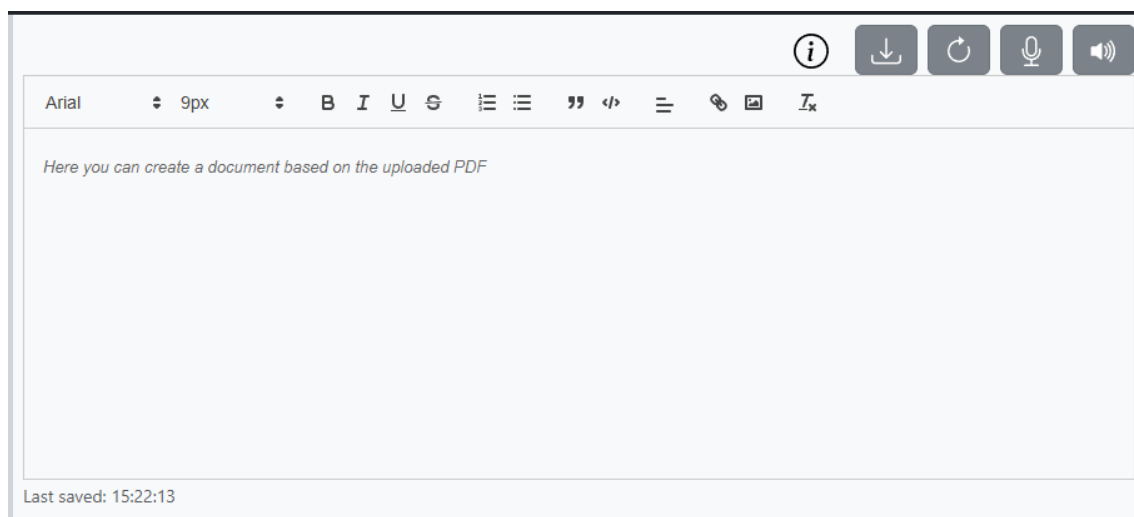


Figure 38: User Interface: `DocumentEditor`

5.5.1. Component Overview

The following simplified UML class diagram (see Figure 39) provides an overview of the `DocumentEditor` component, highlighting its attributes, methods, and key dependencies. It maintains various state variables to track the document's content, modification status, and user interactions.

Key attributes include `editorState`, which stores the current document state, and `hasChanges`, a flag indicating whether there are unsaved modifications. The `lastSavedContent`

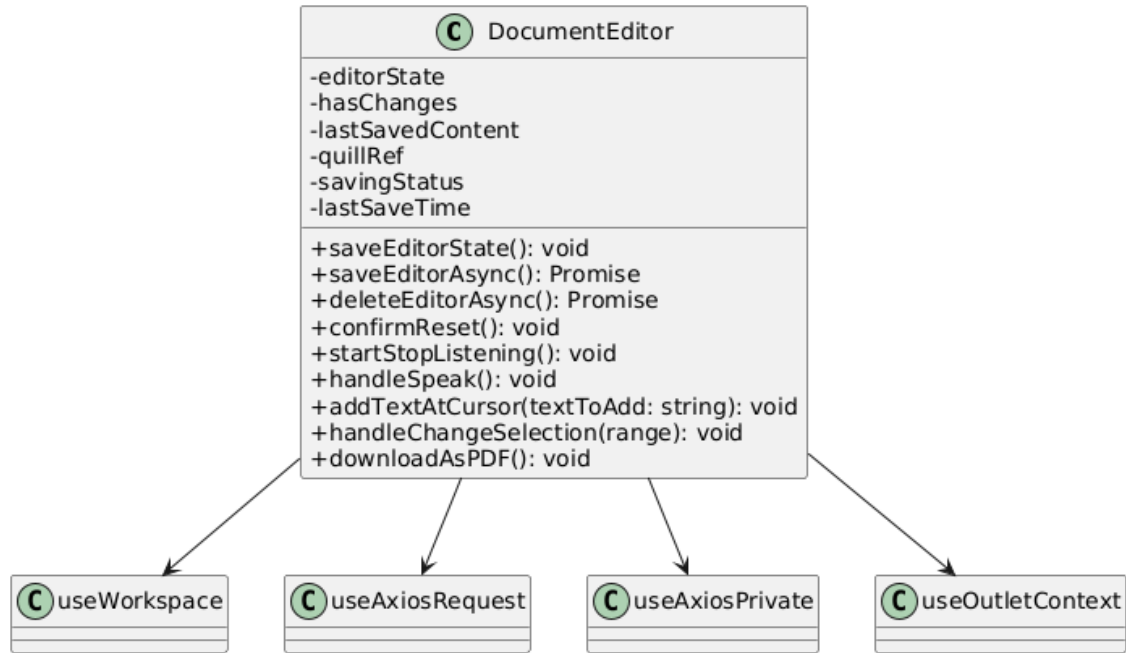


Figure 39: UML: DocumentEditor Class

attribute helps track the last saved version, ensuring the autosave functionality. Additionally, `savingStatus` and `lastSaveTime` provide users with feedback on the document's save state. This information is displayed below the editor as a small badge, along with a timestamp showing the last save time. Other attributes, such as `quillRef`, maintain a reference to the Quill editor instance (see Section 5.5.2).

To facilitate document management, the **DocumentEditor** also contains several methods, with only the most relevant ones described below. The `saveEditorAsync()` function performs the request to save changes on the document in the backend. The `deleteEditorAsync()` function resets the document content and is triggered via the `confirmReset()` method. The component also includes an export functionality through `downloadAsPDF()`, allowing users to download their document while preserving formatting.

The `addTextAtCursor(textToAdd: string)` method inserts the chat's output into the editor.

Multimodal interaction is supported through speech-based functionalities. The `startStopListening()` function enables speech recognition, converting spoken input into text, while `handleSpeak()` utilizes text-to-speech technology to read out the document's content. Additionally, the component leverages custom hooks (e. g. `useAxiosRequest`) to encapsulate reusable logic and improve the readability of the code by reducing the class size.

5.5.2. Functionality of the DocumentEditor

The DocumentEditor interface is built using the **ReactQuill**¹⁷ library, which provides a rich text editor with a customizable toolbar. During the course of the project, multiple different rich text editor libraries were investigated and tested. However, many of them had significant drawbacks, such as lacking recent updates and support, being insufficiently flexible or customizable, or having specific issues—such as problems handling comboboxes properly. After thorough evaluation, ReactQuill was chosen as the most suitable option, offering the necessary features for our requirements. The final toolbar in TextVision’s editor includes font selection, text formatting (bold, italic, underline, strikethrough), lists (ordered and unordered), blockquotes, code blocks, text alignment, as well as support for links and images. The editor also supports custom font sizes and font families, which are defined using the formatting modules. Sourcecode 6 provides an overview of these formats and the toolbar configuration supported by TextVision.

```
1  const modules = {
2    toolbar: {
3      container: [
4        [
5          { font: ['Arial', 'Georgia', 'Courier', 'Roboto', 'Times', '
              Verdana'] },
6          { size: ['9px', '10px', '11px', '12px', '14px', '16px', '18px
              ', '20px', '22px', '24px', '26px', '28px'] }
7        ],
8        ['bold', 'italic', 'underline', 'strike'],
9        [{ list: 'ordered' }, { list: 'bullet' }],
10       ['blockquote', 'code-block'],
11       [{ align: [] }],
12       ['link', 'image'],
13       ['clean']
14     ]
15   }
16 };
17
18 const formats = [
19   'font', 'size', 'bold', 'italic', 'underline', 'strike',
20   'list', 'bullet', 'blockquote', 'code-block', 'align',
21   'link', 'image'
22 ];
```

Sourcecode 6: ReactQuill Toolbar Configuration

¹⁷React-Quill. (n.d.). Npm. Retrieved February 11, 2025, from <https://www.npmjs.com/package/react-quill>

The editor's content is stored and managed with the `editorState` variable. The editor's state is automatically saved every 30 seconds or when the user attempts to close or refresh the page. This ensures that the user's work is not lost due to unexpected interruptions. After saving, the latest state is automatically fetched from the server and displayed. The save functionality is implemented using the `Save EditorState` endpoint, as described in Section 4.1.4.

5.5.3. Speech-to-Text and Text-to-Speech

To enhance accessibility and usability, the `DocumentEditor` includes speech-to-text and text-to-speech functionalities. Users can activate both of these features by clicking the corresponding speech buttons in the toolbar.

The speech-to-text feature allows users to dictate text into the editor using their microphone. This is implemented using the `useSpeechToText` hook, which uses the **Web Speech API**¹⁸ to convert spoken words into text. The transcribed text is automatically inserted at the current cursor position in the editor.

The text-to-speech feature allows users to have the editor's content read aloud. This is implemented using the `useTextToSpeech` hook, which also utilizes the Web Speech API. The whole multimodality concept, including these features, is described in more detail in Section 5.7.

5.5.4. Exporting Documents as PDF

The `DocumentEditor` allows users to export their documents as PDFs. This functionality is implemented using the `pdfmake`¹⁹ library, which converts the editor's HTML content into a PDF document. The conversion process preserves the formatting and styling applied in the editor, including fonts, sizes, alignment, and lists. The PDF is generated with A4 page size and standard margins (72 points on all sides). Users can download the PDF by clicking the "Download as PDF" button in the toolbar. The following code snippet illustrates the PDF generation process:

```
1 const downloadAsPDF = () => {
2   const quillEditor = quillRef.current?.getEditor();
3   if (!quillEditor) return;
4
5   const htmlContent = editorState.state;
6   const pdfMakeContent = htmlToPdfMake(htmlContent);
7
8   const docDefinition = {
9     content: pdfMakeContent,
```

¹⁸Web Speech API - Web APIs — MDN. (2023, February 19). Mozilla Developer Network. Retrieved February 14, 2025, from https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

¹⁹pdfmake. (n.d.). Retrieved February 9, 2025, from <https://pdfmake.github.io/docs/0.1/>

```
10 styles: {
11   'ql-align-left': { alignment: 'left' },
12   'ql-align-center': { alignment: 'center' },
13   'ql-align-right': { alignment: 'right' },
14   'ql-align-justify': { alignment: 'justify' },
15   'ql-size-9px': { fontSize: 9 },
16   'ql-size-10px': { fontSize: 10 },
17   // Additional styles for other font sizes...
18 },
19 pageMargins: [72, 72, 72, 72],
20 pageSize: 'A4',
21 };
22
23 pdfMake.createPdf(docDefinition).download('
24   TextVision_Document.pdf');
```

Sourcecode 7: PDF Generation

5.5.5. Resetting the Editor

Users can reset the editor to its initial state, effectively clearing all content. This is achieved by sending an HTTP POST request to the `Delete EditorState` endpoint (see Section 4.1.4). The request includes the current `workspace` ID, and the server deletes the saved editor state for that workspace. Upon successful deletion, the frontend updates the editor to reflect the reset state. A confirmation modal is displayed to ensure that users do not accidentally reset the editor. The reset functionality allows users start fresh or remove outdated content without having to create a new workspace.

5.5.6. Integration with Chat Functionality

The `DocumentEditor` integrates with the chat functionality (see Section 5.6) to allow users to insert generated text directly into their documents. When a user receives a response from the chat, they can click an icon next to the response to insert the text at the current cursor position in the editor. The inserted text is highlighted briefly to draw the user's attention and then integrated into the document. The goal of this feature is to enhance the workflow by enabling users to quickly incorporate chat output into their editor's document, without the need for manual copying and pasting.

5.6. Chat

Figure 48 depicts the Chat interface, which is divided into three primary sections. The upper section, known as the `StickyChatHeader`, displays saved prompts in a drop-down

menu, allows selection of the LLM model, provides an option to include marked PDF notes as context, and shows an icon indicating the WebSocket connection status. The central section, the `ChatList`, presents the chat history, while the lower section, the `ChatInputForm`, facilitates prompt creation for the LLM and supports the addition of images before sending the completed prompt.

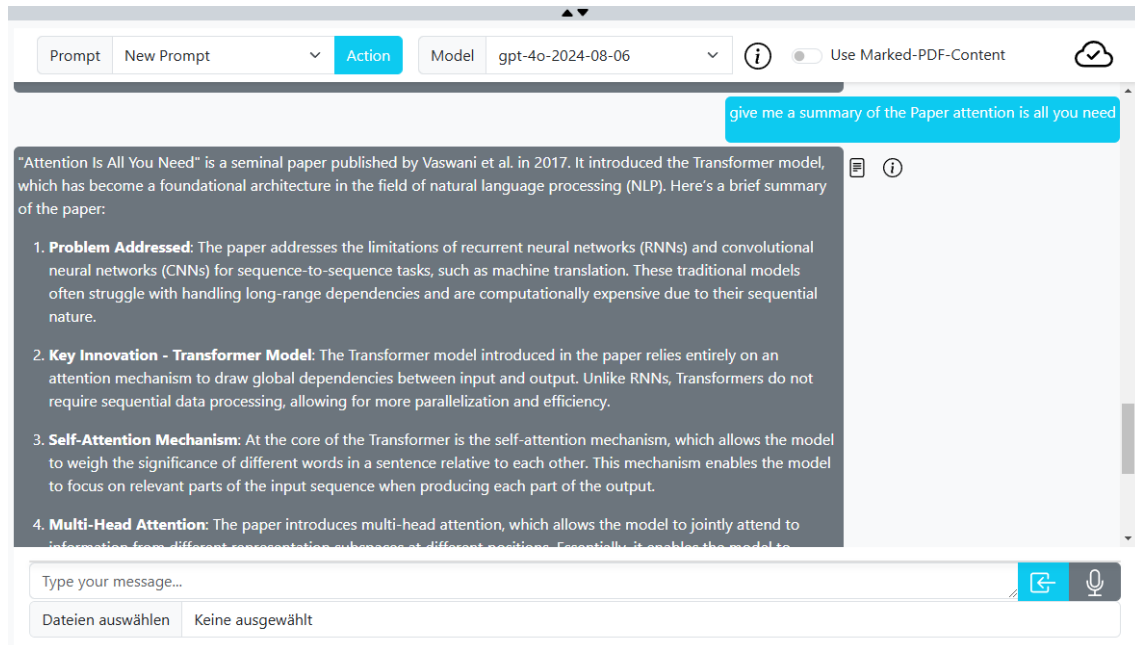


Figure 40: Chat interface

5.6.1. Structure and Functionality

This subsequent chapter delineates the structural organization of the chat interface components and illustrates their respective functionalities.

ChatComponent

Internally, the Chat interface is managed by the `ChatComponent`, as illustrated in Figure 41, which incorporates the aforementioned three components (`StickyChatHeader`, `ChatList`, `ChatInputForm`). The principal responsibilities of the `ChatComponent` include managing the WebSocket instance, propagating state changes to its child components, and interacting with the Backend via HTTP to retrieve stored chat messages.

As described in Section 2.3.4, the establishment of a WebSocket connection necessitates completing an opening handshake. However, the handshake does not permit the inclusion of additional headers, rendering it impossible to perform authentication during the

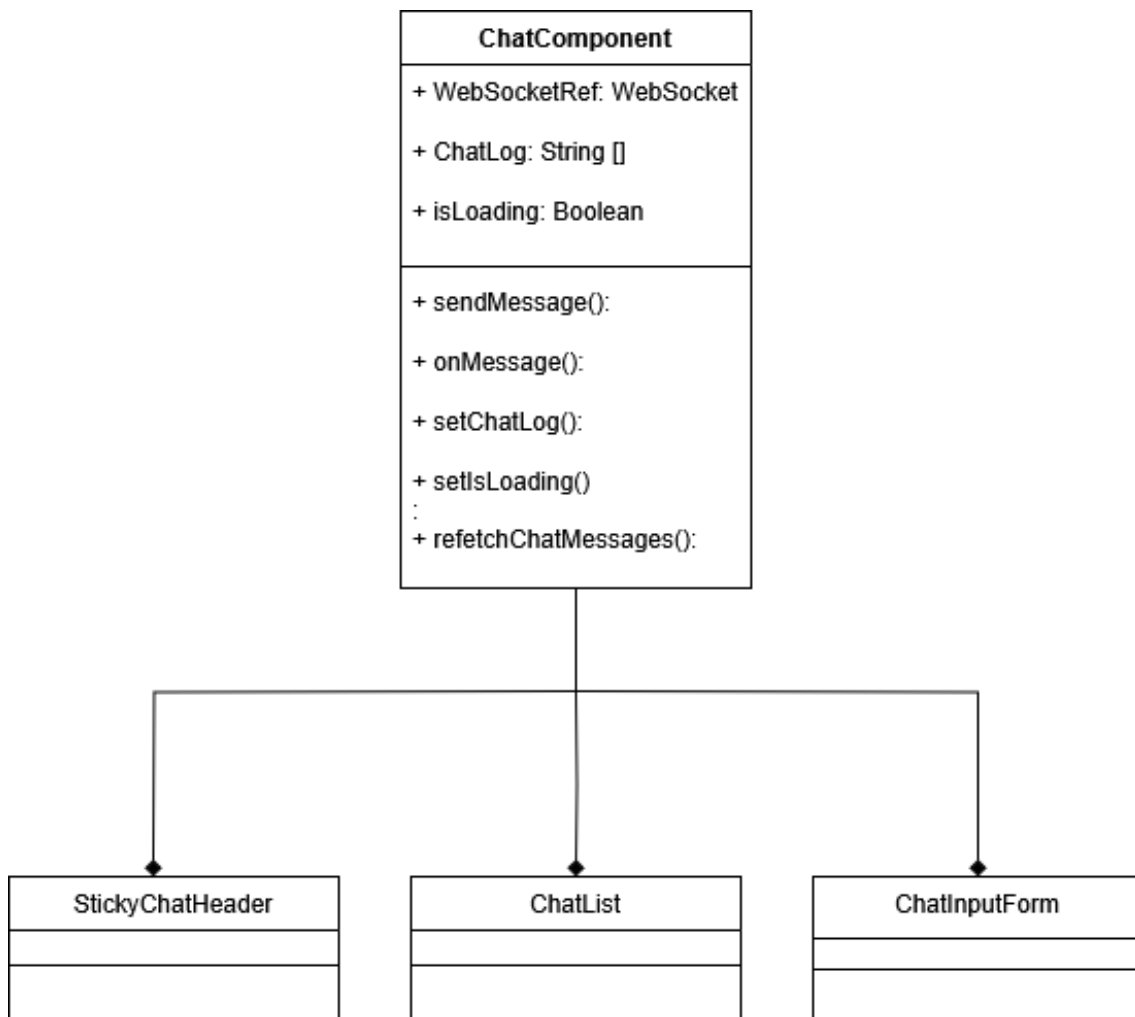


Figure 41: Simplified UML Diagram of the ChatComponent

handshake phase. To maintain the requirement of authenticated access (see Section 5.2.5), modifications were implemented. As shown in Sequence Diagram 42, the authentication process for the WebSocket connection proceeds as follows: a WebSocket connection is initiated only if an authentication access token is available for the user. Following the initiation, the WebSocket's opening handshake is completed. An event listener monitors the 'open' event of the WebSocket connection, which is triggered upon successful connection establishment. This event prompts the Frontend to transmit a message to the Backend via the WebSocket connection; this message includes an action field with the value 'auth' and the user's stored access token. Upon successful authentication on the Backend, an empty message is returned. Receipt of this response signifies that the WebSocket connection has been successfully authenticated, and the WebSocket status is subsequently updated to 'Connected'.

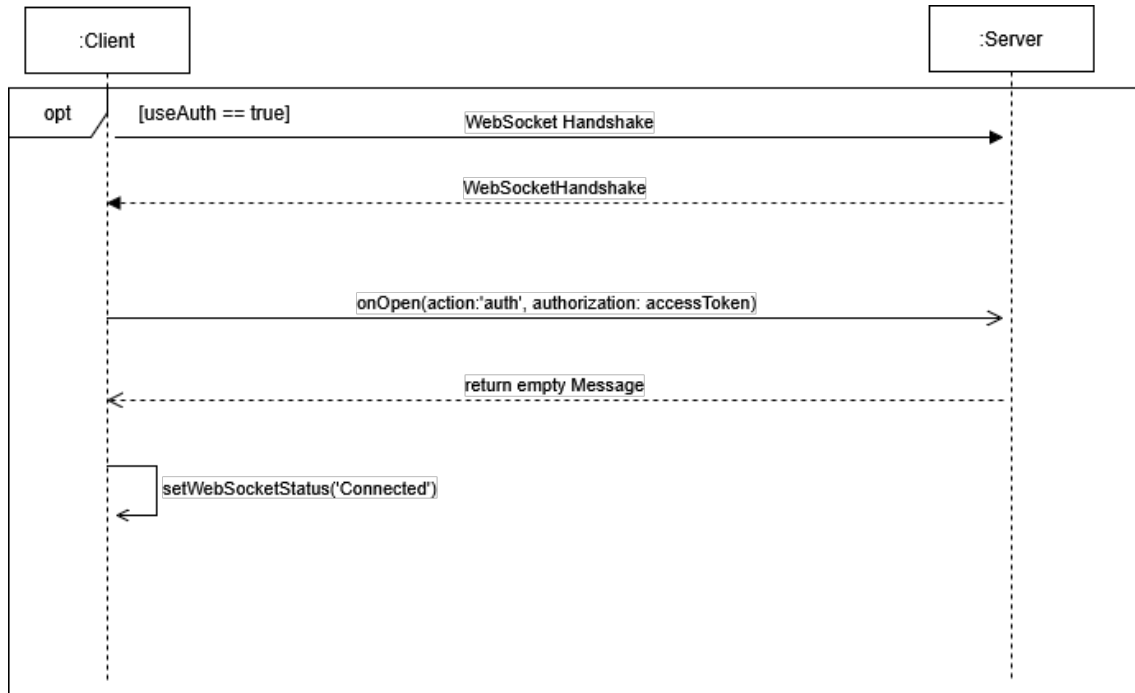


Figure 42: Sequence Diagram of the authenticated WebSocket connection

The management of the established WebSocket connection primarily involves two operations: sending and receiving messages. Sourcecode 8 illustrates the structure of a chat message transmitted from the Frontend. This message includes an action field with the value 'chat' to indicate that it is a chat message, a message field containing the user's formulated prompt as a string, and an audio field consisting of a WAV audio blob. Additionally, an image field is provided, which may be empty, contain a single image, or include multiple images represented in base64 encoding. The message also contains a `workspaceId` to enable the Backend to identify the corresponding workspace, a `pdfId` corresponding to the currently displayed PDF (with a null value if no PDF is displayed), and a `modelId` specifying the selected LLM model intended to generate the response.

```

1 websocketRef.current.send(
2     JSON.stringify({
3         action: 'chat',
4         message: message,
5         audio: base64Audio,
6         images: images ? [...images] : [],
7         workspaceId: workspace,
8         pdfId: pdfId ? pdfId : null,
9         modelId: selectedModelId,
  
```

```
10         keyword: keyword,
11     }},
12 );
```

Sourcecode 8: Chat message

After the message is sent, the state variable `isLoading` is set to true, triggering the display of a loading message to indicate that the LLM is generating a response. Upon receiving the response from the LLM, the `onMessage` function is invoked. This function removes the loading message and updates the `chatLog` state, thereby prompting the `ChatList` component to re-render and incorporate the new chat message into the chat history. The final functionality implemented within the `ChatComponent` involves retrieving the chat history. To accomplish this, the system utilizes an authenticated `axiosPrivate` instance to ensure secure communication. An HTTP POST request is then issued to the endpoint `"api/textVision/chat/chatHistory"`, which returns a list of chat messages

ChatInputForm

The `ChatInputForm` component provides users with a multimedia interface for generating prompts. The primary method involves typing the prompt into the provided text field, with the option to attach images via a file uploader. Alternatively, users may activate the audio input functionality by pressing the 'Microphone' button located on the right side of the `ChatInputForm`. Activating this button initiates an audio listening process—assuming a connected microphone is available—allowing the user to dictate the prompt. A subsequent press of the button terminates the audio capture, after which the audio prompt is transmitted via the WebSocket connection.

Furthermore, the `ChatInputForm` incorporates a prompt suggestion feature. Before its implementation, several considerations were evaluated. The first aspect addressed the methodology for generating suggested prompts. One approach involved performing a prefix check on the user's current input against a set of predefined prompts, which could include the most commonly used phrases. This method offers notable benefits, such as real-time responsiveness—allowing suggestions to be generated and displayed concurrently with user input—and alignment with established practices observed in platforms like Google²⁰, YouTube²¹, and OpenAI²², thereby leveraging user familiarity. However, a limitation of this approach is that the suggestions are not customized to the user's specific context, which may occasionally reduce their relevance.

An alternative approach considered the generation of suggestions using LLMs that incorporate contextual information, such as the currently displayed PDF and the historical chat messages. This method has the advantage of producing context-specific suggestions tailored to the user's needs. Nonetheless, it also introduces potential drawbacks, including

²⁰<https://www.google.com/>

²¹<https://www.youtube.com/>

²²<https://openai.com/>

increased response times due to the processing overhead of contextual analysis. Additionally, careful consideration was required regarding the timing of requests; since each request prompts the LLM to generate a response, sending a request for every character input would lead to excessive computational costs.

Due to TextVision’s architecture, which permits the use of both a PDF and chat history as contextual inputs, the second approach—employing an LLM to generate tailored prompt suggestions—was implemented. This decision was based on the value placed on producing more contextually appropriate prompts, despite the potential increase in response time. Figure 43 illustrates a comparison between prompt suggestions from OpenAI²³, which operates similarly to the first approach, and those from TextVision. In this example, the paper “Attention is All You Need” (Vaswani et al., 2017) was used as the displayed content.

The comparison indicates that prompt suggestions generated by TextVision are more closely aligned with the content of the displayed PDF. However, this method results in longer response times, and the suggested prompts do not begin with the input fragment “What is,” which may cause some user confusion regarding the rationale behind the selected suggestions. The decision to allow the LLM the freedom to generate suggestions without forcing the inclusion of the user’s initial input was deliberate, with the goal of enhancing the originality and relevance of the prompt suggestions.

The mechanism implemented for handling prompt suggestion requests is illustrated in the accompanying State Diagram 44. When the user enters text into the input field, the system first verifies whether the text length exceeds three characters. If this condition is met and no additional text input is detected for a period of two seconds, an HTTP POST request is dispatched to the endpoint “`api/textVision/chat/promptSuggestions`” to retrieve prompt suggestions. The function responsible for initiating this HTTP request is debounced, such that it is executed only after a fixed delay of 2000 milliseconds; any additional keystrokes within this interval reset the timer.

Upon dispatching the request, a prompt suggestion accompanied by a loading animation is displayed to indicate that suggestions are being generated. If the user provides new input while the loading suggestion is visible, the loading suggestion is immediately removed, and no prompt suggestions are displayed, returning the system to the text insertion state. In the absence of further input, three prompt suggestions are eventually presented to the user (see Figure 43).

Should the user select one of these suggestions, the chosen prompt is inserted into the text field, and the suggestion display is dismissed. Conversely, if the user continues to input text, the system reverts to the text insertion state, and any displayed suggestions are cleared. Additionally, to prevent unnecessary requests after a prompt has been submitted, the act of sending a prompt cancels any pending debounced function calls.

²³<https://openai.com/>

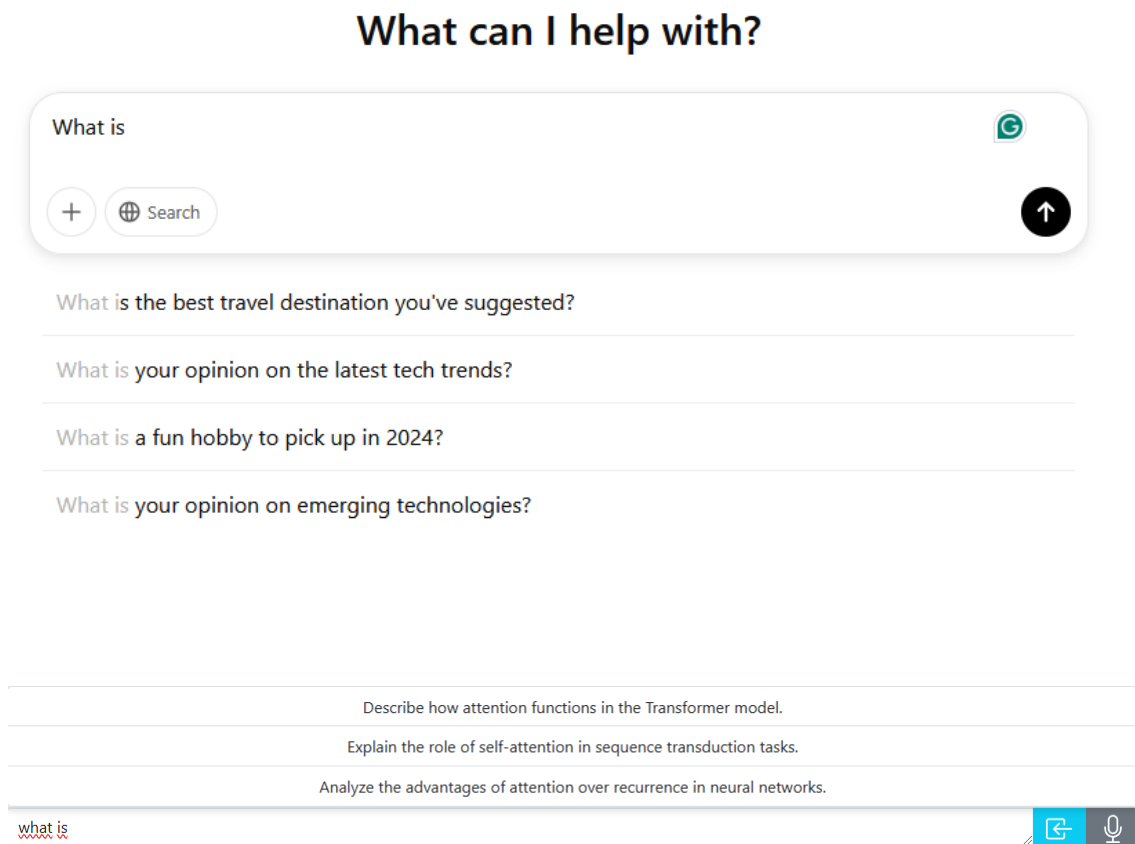


Figure 43: Comparison of OpenAI(top) and TextVision(bottom) prompt suggestions

ChatList

The **ChatList** component is responsible for displaying the chat history with context-dependent styling. For messages originating from the user, the component aligns the message to the right and applies a light blue background color. In contrast, messages from the LLM are aligned to the left with a light gray background. In cases where a Backend error occurs or the WebSocket connection is nonfunctional, a placeholder error message is centrally displayed with a red background (see Figure 45).

To facilitate the rendering of markdown in LLM responses, the **react-markdown**²⁴ library was integrated (see Figure 48 the text in the response message is enumerated and the starting points are highlighted). Considering that one potential application of TextVision is to assist scientific researchers, the inclusion of LaTeX rendering for mathematical formulas was evaluated. However, since the primary objective of TextVision is to support document creation, and the **DocumentEditor** would also need to render LaTeX for consistent functionality, this feature was deemed too resource-intensive to implement and was therefore omitted.

²⁴<https://remarkjs.github.io/react-markdown/>

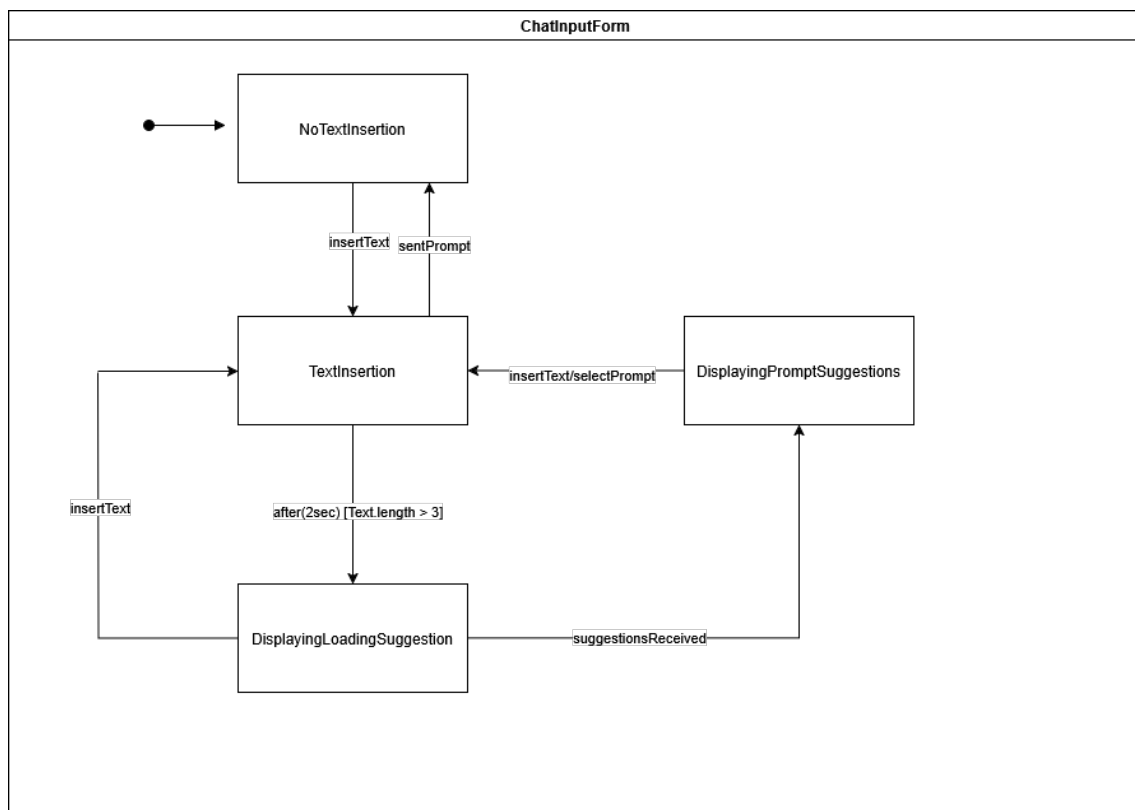


Figure 44: State Diagram of the prompt suggestion lifecycle

If the user employs the audio function to send a message, the resulting chat message is displayed with an integrated audio player. This player facilitates standard controls, including the initiation and termination of playback, volume adjustment, and fast-forwarding (see Figure 45).

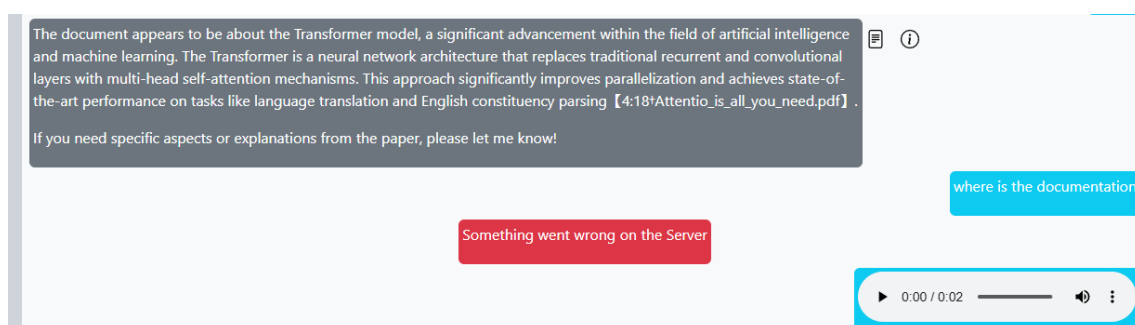


Figure 45: All possible message forms

5.6.2. Interaction with PDF Viewer and DocumentEditor

This section illustrates the interaction between the Chat component and the two primary components, PDF Viewer and DocumentEditor. It discusses the implemented interaction mechanisms and the rationale underlying these design choices.

Interaction with PDF Viewer

The **PDF Viewer** and Chat components facilitate bidirectional interaction, as depicted in Sequence Diagram 46. The first mode of interaction occurs when a user selects content within the PDF and activates the "ask" button. In this instance, the selected text, image, and associated description are formatted using Markdown, resulting in an input structure that includes "MarkedText," "MarkedImage", and a "task" description. This formatted input is then transmitted to the Backend in the standard message format (see Sourcecode 8), whereupon a response from the LLM is anticipated.

The second mode of interaction is initiated when the user designates a note as context. Notably, this action involves dispatching a request to the server with parameters including `noteId`, `pdfId`, and `newMarkedStatus`, to update the corresponding record in the database. Since the server serves as the SSOT, it subsequently informs the **PDF Viewer** of the status change for the marked note. The **PDF Viewer** then relays this updated status to the **ChatComponent**. Although these intermediary steps are omitted in the simplified Diagram 46, they are essential for maintaining data integrity. When the user opts to utilize the marked PDF content as context by selecting the **MarkedPdfContent** slider, they may then input a prompt. Upon submission, the marked PDF content is formatted in Markdown—comprising `MarkedText`, `MarkedImages`, and the task (i.e., the entered prompt)—and sent to the Backend in the usual message format (see Sourcecode 8), with an LLM response subsequently expected.

Interaction with DocumentEditor

The interaction between the **DocumentEditor** and the Chat is designed to enable the insertion of LLM responses into the **DocumentEditor** via two distinct mechanisms.

The primary method involves a button located at the top-right corner of each LLM response. When this button is pressed, the entire response is inserted into the document. Recognizing that LLM responses can be extensive and may contain irrelevant content, a secondary method was implemented. In this alternative approach, users may highlight a portion of the LLM response and press the key combination Ctrl plus Enter to insert only the selected text into the **DocumentEditor**. This method circumvents the inconvenience of scrolling to the top of a lengthy message to press the insertion button and then scrolling back to the desired selection.

An implementation consideration for the secondary method is that the `SelectionAPI`²⁵, which is employed to retrieve the selected text, operates on the entire window or document. However, the functionality to insert the marked text into the **DocumentEditor** is intended to be available solely within the Chat context. Sourcecode 9 illustrates the

²⁵<https://developer.mozilla.org/en-US/docs/Web/API/Selection>

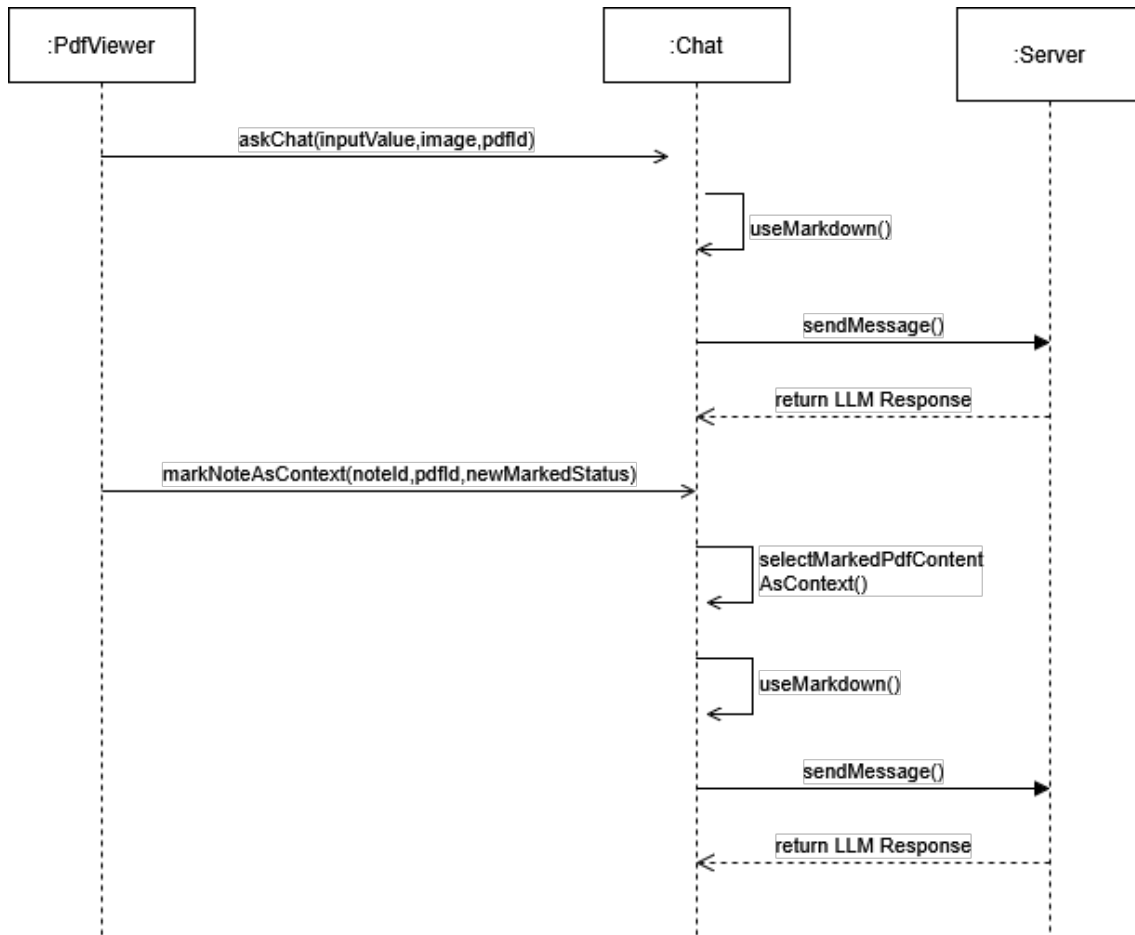


Figure 46: Simplified sequence diagram of PDF Viewer and Chat interaction

implementation that meets these requirements. Initially, the code verifies that the key event corresponds to the Ctrl plus Enter combination. If so, the SelectionAPI is used to retrieve the current text selection from the window. This text is then sanitized by removing any leading or trailing whitespace. If the resultant string is empty, the operation is aborted. Otherwise, the code determines the location of the selection and identifies the corresponding ancestor container. It then verifies whether this ancestor container is within the **ChatContainer**. If this condition is met, the selected text is assigned to the **ServerResponseText**, thereby triggering its insertion into the **DocumentEditor**. Since the selection is evaluated across the entire window, the event listener is attached to the window object.

```

1 useEffect(() => {
2   const handleKeyDown = (e) => {
3     if (e.key === 'Enter' && e.ctrlKey) {

```

```

4      const selection = window.getSelection();
5      const selectedText = selection.toString().trim();
6
7      if (selectedText === '') return;
8
9      // Check if the selection is within the chat
      container
10     const chatContainer = chatRef.current;
11     if (chatContainer && selection.rangeCount > 0) {
12         const range = selection.getRangeAt(0);
13         const { commonAncestorContainer } = range;
14         const isInside = chatContainer.contains(
            commonAncestorContainer);
15         if (isInside) {
16             setServerResponseText(selectedText);
17         }
18     }
19 }
20 };
21
22 window.addEventListener('keydown', handleKeyDown);
23
24 return () => {
25     window.removeEventListener('keydown', handleKeyDown);
26 };
27 }, []);

```

Sourcecode 9: SelectionAPI

5.7. Multimodality

Multimodal functionality is an important part of TextVision, which aims to improve user experience and work efficiency through multiple input and output methods. This section will introduce the implementation and integration of multimodal functionality in detail, including speech-to-text, text-to-speech, chat voice input, and voice commands. These speech-to-text and text-to-speech are integrated with DocumentEditor, chat voice input integrated with chat, while voice commands exist independently and are triggered by trigger words, which provides users with a richer way of interaction.

5.7.1. Speech-to-Text combined with the DocumentEditor

The speech-to-text function allows users to directly convert speech into text through voice input, and insert it into the current cursor position in the DocumentEditor component to

display it. This feature is activated by clicking the microphone icon button of the DocumentEditor. After activation, a strikethrough is added to the microphone icon to indicate that it cannot be clicked again. The recognition will automatically stop 1 second after the user stops speaking. The integration into the DocumentEditor's layout is presented in Figure 47, and the microphone icon button is presented in the upper right corner.

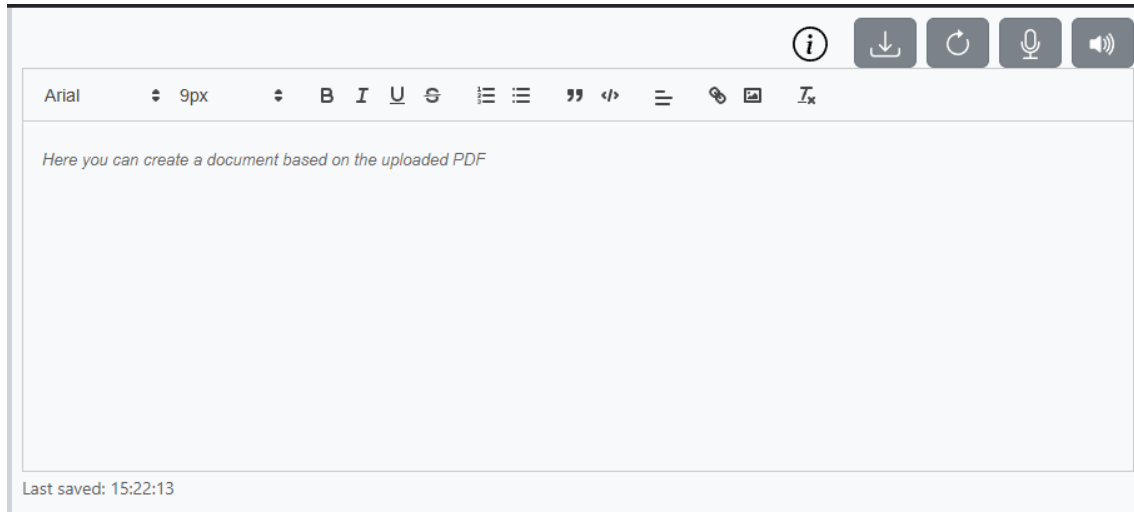


Figure 47: User Interface: DocumentEditor

The Web Speech API is an experimental technology that enables Web applications to process speech data, including speech recognition and speech synthesis. This API was developed to provide speech interaction capabilities for Web applications. Its core functions include:

- **SpeechRecognition interface:** Processes user voice input and converts it into text. Users can configure parameters such as the recognition language and whether to enable continuous recognition. Recognition results are obtained through event listeners (Mozilla Developer Network, 2024).
- **SpeechSynthesis interface:** Converts text into spoken output. Users can control parameters such as speaking rate, pitch, and volume, as well as select different speech synthesis voices (Mozilla Developer Network, 2024).

In the DocumentEditor, speech recognition is implemented using the ‘webkitSpeechRecognition’ interface, which is part of the Web Speech API supported by Webkit-based browsers (an open-source browser engine) such as Chrome and Safari (Mozilla Developer Network, 2024, WebKit, 2025). The configuration of speech recognition includes:

- **continuous:** set to true, indicating continuous recognition (see Sourcecode 10).

- `interimResults`: set to `true`, indicating that interim results are provided (see Sourcecode 10).
- `lang`: set to `'en-US'`, indicating that the recognized language is american english (see Sourcecode 10).

The processing flow of Speech-to-Text voice input includes:

First, the user click the microphone icon button of the `DocumentEditor` component to activate Speech-to-Text recognition via the `startSpeechRecognition` function (see Sourcecode 10). As the user speaks, the spoken words are converted into text and displayed in real-time in the `DocumentEditor`. Punctuation can be added by speaking "comma", "full stop", or "semicolon". This involves the `onResult` callback function which captures the converted text and inserts it into the current cursor position of `DocumentEditor` using the `insertText` function (see Sourcecode 10). The speech recognition will stop automatically, after the user stops speaking for one second. As a result the content of the `DocumentEditor` gets updated, and the microphone icon returns to its inactive state.

```
1  const startSpeechRecognition = () => {
2  const recognition = new webkitSpeechRecognition();
3  recognition.continuous = true;
4  recognition.interimResults = true;
5  recognition.lang = 'en-US';
6  recognition.onstart = () => {
7  };
8  recognition.onresult = (event) => {
9  const transcript = event.results[event.resultIndex][0].
    transcript;
10 insertText(transcript);
11 };
12 recognition.onend = () => {};
13 recognition.start();
14 };
```

Sourcecode 10: Speech-to-Text Implementation

5.7.2. Text-to-Speech combined with the `DocumentEditor`

The text-to-speech feature allows users to convert text content of the `DocumentEditor` component into speech output. This feature is activated by clicking the speaker icon button of the `DocumentEditor`. If the current `DocumentEditor` has no content, it will read aloud the prompt "Current content is empty". The layout of the `DocumentEditor` is presented in Figure 47, and the speaker icon button is presented in the upper right corner. Similar to the previous section about Speech-to-Text, Text-to-Speech also uses Web Speech API to convert text into audio output. The processing flow of text-to-speech output is the following:

The user clicks the speaker icon button of the DocumentEditor to activate text-to-speech output via the `handleSpeak` function (see Sourcecode 11). Then, the system will retrieve the current text content from the DocumentEditor. If the content is empty, it uses the default output "Current content is empty." The `speakText` function creates a `SpeechSynthesisUtterance` instance, passes the text content to it, and starts speech output using the `speechSynthesis.speak`.

speech (see Sourcecode 11). If the audio output is completed, the speech synthesis instance is cleaned up.

```
1  const handleSpeak = () => {
2    if (quillRef.current) {
3      const editorInstance = quillRef.current.getEditor();
4      const editorContent = editorInstance.getText();
5      speakText(editorContent);
6    } else {
7      console.error('Editor reference is null. ');
8    }
9  };
10 const speakText = (text) => {
11   if (!text.trim()) {
12     text = 'Current content is empty. ';
13   }
14   if (isSpeaking) {
15     window.speechSynthesis.cancel();
16     setIsSpeaking(false);
17     return;
18   }
19   const utterance = new SpeechSynthesisUtterance(text);
20   utterance.onstart = () => setIsSpeaking(true);
21   utterance.onend = () => setIsSpeaking(false);
22   window.speechSynthesis.speak(utterance);
23   };
```

Sourcecode 11: Text-to-Speech Implementation

5.7.3. Voice Input combined with the Chat

The chat voice input function allows users to send messages to the chat via speaking. Clicking the microphone icon button of the chat to start this feature. The voice message format will be converted to WAV through the WavEncoder API, and sent to the AI-Backend after Base64 encoding to obtain the message, which will appear in the chat component for users to view. The layout of the chat can be seen in Figure 48 and the microphone icon button is at the bottom right corner of it.

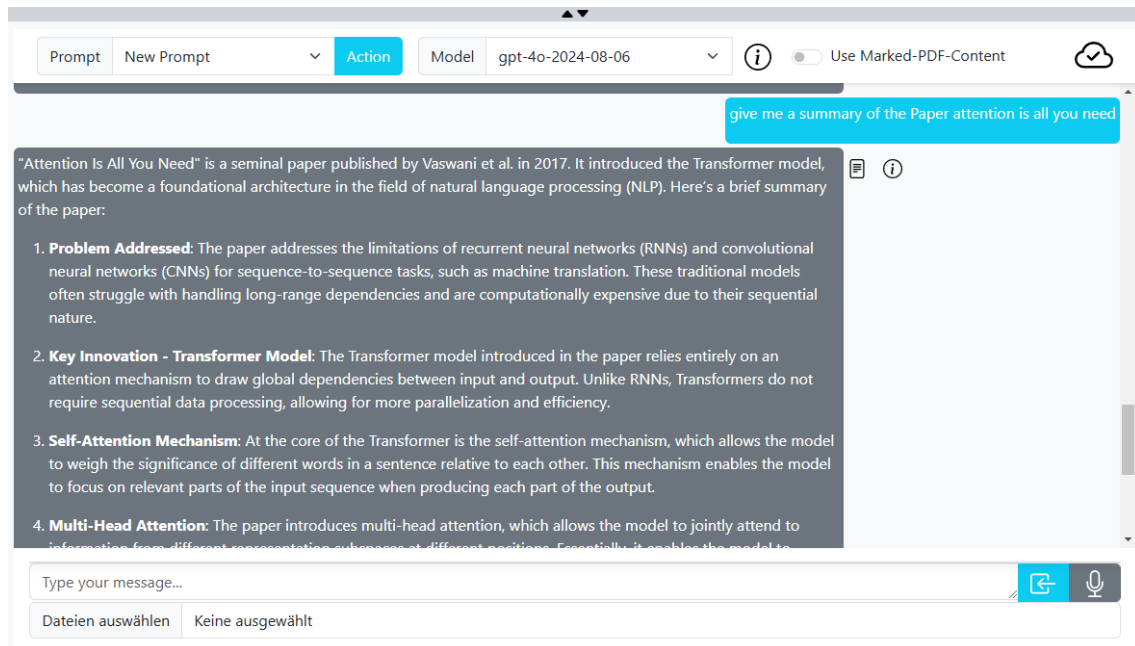


Figure 48: Chat interface

The chat voice function is implemented based on the MediaRecorder API. The MediaRecorder API is a tool for capturing and manipulating media streams, providing functionality for audio recording. In this project, it serves as the foundation for implementing core voice recording features. Voice recording is initiated by calling the `navigator.mediaDevices.getUserMedia` interface, which is the entry point for accessing media devices such as microphones, gaining access to the microphone. This interface returns a Promise that resolves to a media stream once the user grants permission. In the ChatComponent, this interface is used to begin the voice recording process, ensuring that the user's voice input is captured (Mozilla Developer Network, 2025) (see Figure 12).

```

1  const startStopListening = () => {
2      navigator.mediaDevices.getUserMedia(constraints)
3      .then(stream => {
4          const recorder = new MediaRecorder(stream);
5          mediaRecorderRef.current = recorder;
6          recorder.ondataavailable = event => {
7              if (event.data.size > 0) {
8                  audioChunksRef.current.push(event.
9                      data);
10             }
11         };
12     });

```

```
11         recorder.onstop = async () => {
12             const webmBlob = new Blob(audioChunksRef.
13                 current, { type: 'audio/webm' });
14             audioChunksRef.current = [];
15             };
16             recorder.start();
17             setIsListening(true);
18         });
19     } else {
20         const recorder = mediaRecorderRef.current;
21         if (recorder) {
22             recorder.stop();
23             setIsListening(false);
24         }
25     };
```

Sourcecode 12: Start/Stop Voice Recording

Once the user grants permission, the MediaRecorder API's `start()` function begins capturing audio data. This data is stored in the "audioChunksRef" array. When the user stops the recording, the API collects these audio chunks and converts them into a WebM format Blob, a binary storage format for media data (see Sourcecode 12).

When the user stops recording by clicking the chat microphone voice icon button the second time, the `stop()` function of the MediaRecorder API is called and the collected audio chunks will be converted into a WebM Blob, a binary storage format for media data (binary large object) (see Sourcecode 12).

To ensure compatibility with our backend processing, the WavEncoder library is employed to transform this WebM Blob into a WAV format Blob. WAV is a widely recognized audio format that simplifies further processing. The encoded WAV data is then sent to the backend using WebSockets. This transfer is optimized for network transmission by converting the binary audio data into a Base64 string. Base64 encoding converts complex binary data into a text-based format, ensuring safe and efficient network communication. The backend receives this data and can process it further for analysis or storage. This entire flow — capturing audio input, converting it into manageable formats, and transmitting it efficiently — ensures a seamless user experience.

5.7.4. Voice Commands

The voice command feature allows users to directly control the application through voice command input. For example, users can start the voice command by saying the word "Text version", then say "chat" to add the selected PDF text content to the dialog box of the chat component, or say "summarize" to send a summary request to the chatbot. The voice command function is implemented using the Web Speech API and, similarly to

the speech functionality, voice commands also use `webkitSpeechRecognition` interface to access the microphone and start speech recognition. Then, the system checks for browser compatibility and initializes the recognition instance with key configurations (see Sourcecode 13).

```

1  const SpeechRecognition = window.SpeechRecognition || window.
    webkitSpeechRecognition;
2  if (!SpeechRecognition) {
3      console.error('This browser does not support the
        SpeechRecognition API. ');
4      return;
5  }

```

Sourcecode 13: Speech Recognition Initialization

The system listens for specific commands after detecting the spoken input "text version". When a command is recognized, corresponding actions are executed, such as navigation to specific pages or processing text (see Sourcecode 14). Figure 49 illustrates the main voice command processing flow within the application. The process starts with the user providing a voice input, which is then checked for the presence of an activation speech input. If such an activation input is detected, the system proceeds to recognize and process the specific command. Once a command is identified, the corresponding action is executed, and feedback is provided to the user.

```

1  recognitionInstance.onresult = (event) => {
2      const currentTranscript = event.results[event.resultIndex
        ][0].transcript;
3
4      if (currentTranscript.toLowerCase().includes('text
        version')) {
5          const commands = {
6              'prompt_designer': { target: 'promptDesigner',
                keyword: 'promptDesigner' },
7              'summarize': { target: 'chat', keyword: '
                summarize' },
8              'translate': { target: 'chat', keyword: '
                translate' }
9          };
10         Object.entries(commands).forEach(([command, { target,
                keyword }]) => {
11             if (currentTranscript.toLowerCase().includes(
                command)) {
12                 if (target === 'promptDesigner') {
13                     navigate('/promptDesigner');
14                 } else if (target === 'chat') {

```

```

15         speakText('Processing_${command}...');
16     }
17 }
18 });
19 }
20 };

```

Sourcecode 14: Voice Command Processing

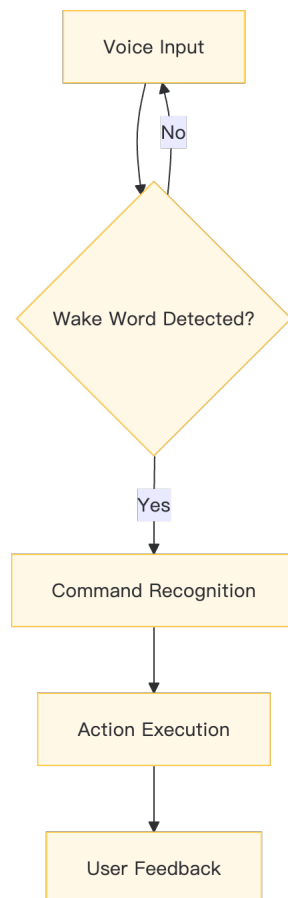


Figure 49: UML Activity Diagram: Voice Command Processing

The `isListening` state is set to "true" when recognition is active and to "false" when it stops (see Sourcecode 15).

```

1  recognitionInstance.onstart = () => {
2      setIsListening(true);
3  };
4  recognitionInstance.onend = () => {

```

```
5   setIsListening(false);  
6 };
```

Sourcecode 15: Speech Recognition End Handler

To provide a high-level overview of the voice command system's architecture and component interactions, Figure 50 presents the Component Interaction Diagram. The Voice Command Context Provider is the central component responsible for managing the voice command functionality. It interacts with the Speech Recognition Component to capture and recognize user speech. The Command Processor is responsible for analyzing the recognized speech and determining the appropriate action to take based on predefined commands. The Page Navigator handles navigation between different pages, such as moving to the Prompt Designer page when the "prompt designer" command is detected. The Chatbot component processes commands related to chat interactions, the Document Editor manages commands related to document editing, and the Prompt Designer handles commands related to designing and managing prompts.

The "Voice Command Context Provider" uses the "Speech Recognition Component" to convert voice input into text. Once the text is recognized, the "Command Processor" checks if the text includes the wake word and any specific commands. If the command is "prompt designer", the "Page Navigator" is triggered to redirect the user to the Prompt-Designer page. The "Chatbot", "Document Editor", and "Prompt Designer" components are contextually activated based on the command detected. The "Prompt Designer" component, when navigated to, allows users to design and manage prompts, which is a separate functionality integrated into the application through this navigation.

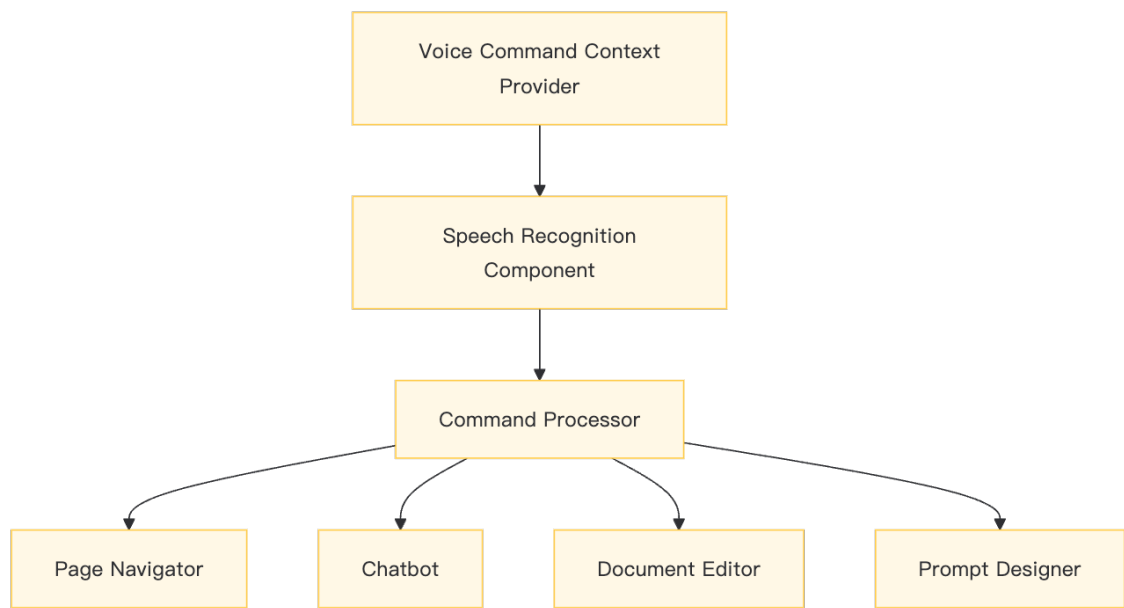


Figure 50: UML Component Diagram: Voice Command Component Interaction

5.8. PromptDesigner

The `PromptDesigner` is a core component of TextVision, designed to help users analyze, optimize, and manage prompts for AI models. This section provides an overview of its functionality, including the evaluation, optimization, and storage of prompts. The `PromptDesigner` communicates with the backend to fetch and store data (AI models, frameworks, prompts) as described in Section 4.1.7.

The user interface of the `PromptDesigner` is designed in a structured and intuitive way. It includes Dropdowns for selecting AI models and Frameworks, Sliders for adjusting parameters, and buttons for triggering evaluation and optimization. The results of these processes are displayed in an organized manner, with tabs for navigating between different issues and their suggested improvements. Additionally, most fields and components in the design also include info icons that users can hover over for additional details.

5.8.1. Component Overview

The `PromptDesigner` component is responsible for managing prompts, including their creation, evaluation, optimization, and deletion. It interacts with various hooks and external components to fetch data, handle user interactions, and display results. The component maintains several state variables to track the current prompt, its evaluation results, and any changes made by the user. Figure 51 provides a simplified UML class diagram that outlines the key attributes and methods of the `PromptDesigner` component.

Key attributes include `localTemplates`, which stores a list of prompts, and `currentTemplate`, which holds the currently selected prompt for editing or evaluation. The com-

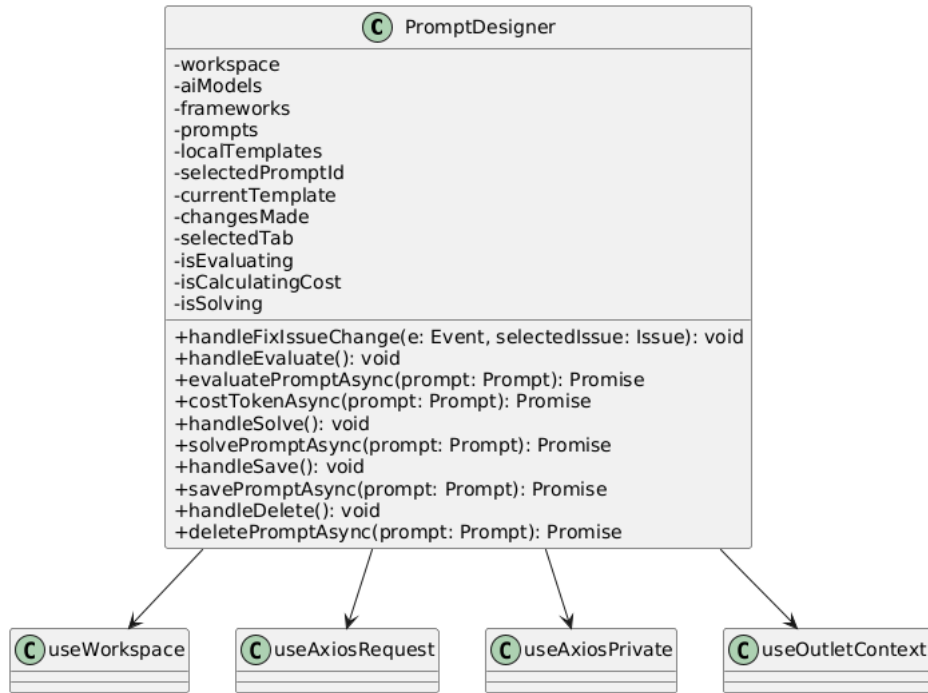


Figure 51: UML: PromptDesigner Class

ponent also tracks changes made to the prompt (`changesMade`) and the selected tab for displaying evaluation results (`selectedTab`).

The component's methods include `handleEvaluate()`, which triggers the evaluation of the current prompt, and `handleSolve()`, which optimizes the prompt based on identified issues. Additionally, `handleSave()` and `handleDelete()` manage the storage and deletion of prompts, respectively.

5.8.2. Functionality of the PromptDesigner

The PromptDesigner interface allows users to create and manage prompts, evaluate their effectiveness, and optimize them based on feedback from an AI model. The interface is divided into several sections, each serving a specific purpose:

Prompt Selection and Creation

In the PromptDesigner's first section (see Figure 52), users can either select an existing prompt from a Dropdown menu (if one has been created previously) or create a new one by choosing the default "New Prompt" option. Additionally, the users can modify the "name", "prompt", "goal", and "description" fields. The prompt and goal fields are particularly important, as they define the desired outcome of the prompt and its specific goal. They are mandatory and guide the evaluation and optimization processes. The goal field was introduced to help users - that aren't familiar with prompt creation - create more

effective prompts. The description field provides additional context, making it easier to understand the prompt’s intent; this could be especially useful when a user revisits and adjusts previous prompts at a later point of time.

PromptDesigner

Select Prompt

Prompt

New Prompt

▼

i

Prompt

Name

New Prompt

i

Prompt

Write your prompt here...

i

Goal

This is where your goal comes in...

i

Description

Optional: This is where your description comes in...

i

Figure 52: User Interface: PromptDesigner (Prompt Section)

Evaluation

Figure 53 showcases the evaluation process, which analyzes the prompt based on a selected AI model (see Section 7.2.1) and Framework (see Section 7.2.3). The user can select a Framework, which is used as a set of rules to evaluate the prompt on. They can also adjust the temperature parameter, which controls the creativity and randomness of the AI’s responses. Upon evaluation, the component displays a rating (on a scale of 1 to 5 stars) and identifies potential issues with the original prompt. Each issue includes a description and a suggested improvement, which users can choose to address during the optimization process by selecting the corresponding checkbox. The evaluation results are displayed in a tabbed interface, where each tab corresponds to a specific issue. The evaluation also provides a cost and token estimate, helping more advanced users understand the computational resources required for executing the prompt.

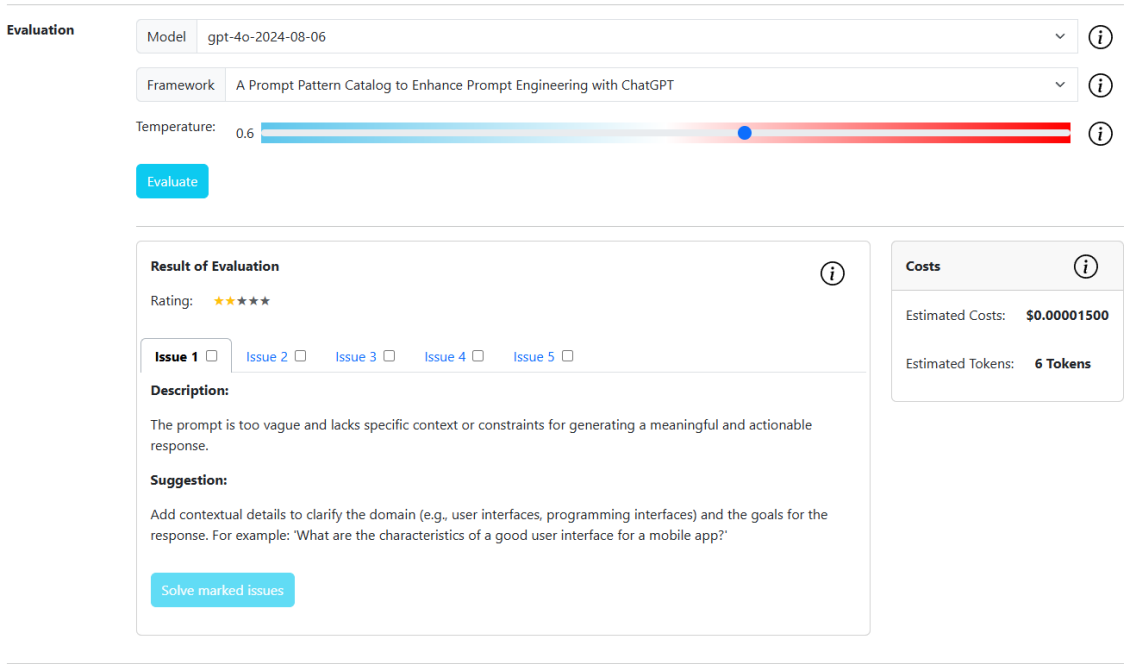


Figure 53: User Interface: PromptDesigner (Evaluation Section)

Optimization

After evaluating the prompt, users can optimize it by selecting specific issues to fix and initiating the optimization. This process generates an improved version of the prompt, which users can apply to replace the original (see Figure 54). This feature ensures that prompts are iteratively refined and aligned with the user’s desired goals; to achieve better results from the chat when using it. Afterward, the user can save the created prompt to the Dropdown menu by clicking the corresponding button. They can then find and select the prompt on the "Home" page (Dropdown at chat component) as well (see Section 3.4).

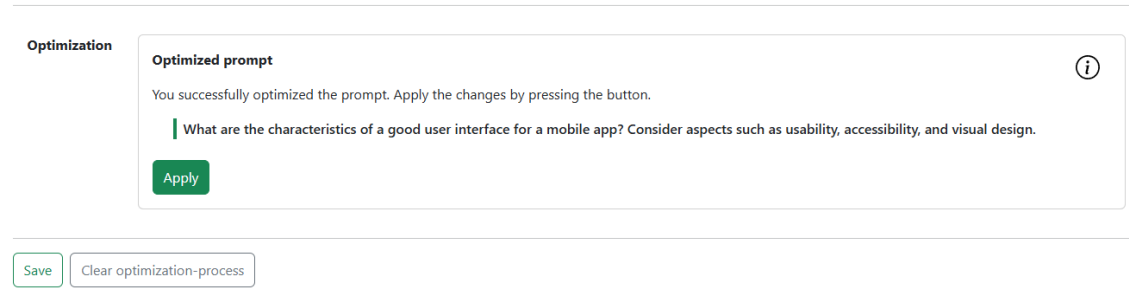


Figure 54: User Interface: PromptDesigner (Optimization Section)

5.8.3. Error Handling and User Feedback

The PromptDesigner includes error handling to ensure a smooth user experience. Errors encountered during Backend requests are generally displayed in a Modal. Additionally, feedback for interactions is provided through toast messages, which notify users of successful actions such as saving or deleting a prompt.

5.9. FAQ

The FAQ page in TextVision serves as a structured knowledge base to help users navigate the platform effectively. It provides categorized answers to frequently asked questions, an integrated search function, and a contact form for direct inquiries. The design ensures that users can quickly find solutions to their questions without the need for external assistance, increasing usability and efficiency.

The FAQ page consists of three primary components as shown in Figure 55:

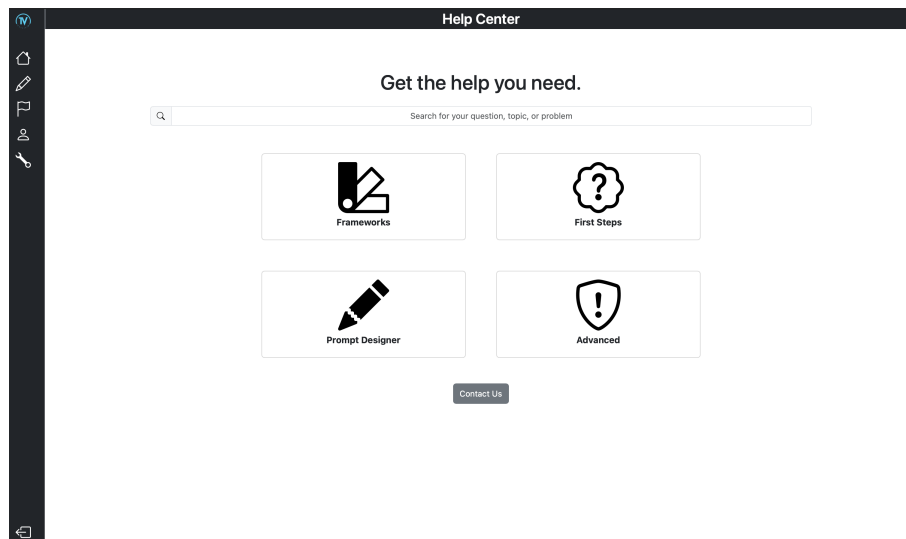


Figure 55: Overview of the FAQ Page

- **Search Bar:** Located at the top of the page, it allows users to enter queries and receive instant suggestions. The search bar is implemented with an autocomplete feature that dynamically filters relevant questions based on user input.
- **Category-based FAQ sections:** Questions are grouped into logically structured sections to improve accessibility. Each section contains expandable questions, allowing users to view answers without excessive scrolling.
- **Contact Form:** For questions not covered in the FAQ, users can submit inquiries via a modal form. This form includes input validation for email addresses and required fields to ensure accurate submission.

Moreover, Figure 56 shows the contact modal. In addition to the overview in Figure

Contact Us

Email address

Topic

Select a topic

Message

0/200 characters

CancelSend Message

Figure 56: Contact Modal of the FAQ Page

55 shows how the FAQ page is divided into four main categories, each covering different aspects of the platform. The table 12 below summarizes the content included:

Table 12: Table of Categories, Descriptions, and Example Questions

Category	Description	Example Question
Frameworks	Covers supported AI evaluation frameworks and best practices for prompt engineering.	What frameworks support evaluation in the PromptDesigner?
First Steps	Guides new users through the onboarding process, including document uploads and workspace management.	How do I upload a PDF file?
PromptDesigner	Provides detailed instructions on using the PromptDesigner to create and optimize AI prompts.	How do I create a new prompt?
Advanced	Addresses technical and troubleshooting aspects such as data storage policies and performance optimizations.	How does the platform handle multimodal input?

5.9.1. Memoization (useMemo) for Efficient Rendering

The `useMemo` hook is used to store and manage FAQ categories efficiently. This prevents unnecessary recalculations and improves the performance of the component.

Implementation

The `useMemo` hook ensures that the `categories` array is not recompiled unless dependencies change. This optimizes rendering by preventing unnecessary recalculations when state updates occur. Furthermore, it improves efficiency by storing static data and reducing computational overhead.

```
1 const categories = useMemo(  
2   () => [  
3     { title: 'Frameworks', icon: frameworksIcon, sectionKey: '  
4       frameworks', link: '/faq/frameworks', questions: [...] },  
5     { title: 'First Steps', icon: firstStepsIcon, sectionKey: '  
6       first-steps', link: '/faq/first-steps', questions: [...] },  
7     { title: 'Prompt Designer', icon: promptDesignerIcon,  
8       sectionKey: 'prompt-designer', link: '/faq/prompt-designer',  
9       questions: [...] },  
10    { title: 'Advanced', icon: troubleshootingIcon, sectionKey: '  
11      Advanced', link: '/faq/Advanced', questions: [...] },  
12  ],  
13  []  
14 );
```

Sourcecode 16: useMemo Hook Implementation

The `useEffect` hook listens for changes in `searchTerm` and dynamically updates `filteredSuggestions`. The filtering process is done asynchronously to ensure smooth UI updates without blocking the main thread. Then `setFilteredSuggestions(filtered.slice(0, 5))` limits the results displayed to improve readability and performance.

```
1 useEffect(() => {  
2   if (searchTerm.trim()) {  
3     const filtered = categories  
4       .flatMap((category) =>  
5         category.questions.map((question) => ({ question, link:  
6           category.link })))  
7     .filter((item) => item.question.toLowerCase().includes(  
8       searchTerm.toLowerCase()));  
9     setFilteredSuggestions(filtered.slice(0, 5));  
10  } else {  
11    setFilteredSuggestions([]);  
12  }  
13 }, [searchTerm, categories]);
```

Sourcecode 17: Dynamic Search Implementation

In terms of accessibility, Keyboard Navigation Support enhances the search experience by allowing users to navigate search results using arrow keys.

```
1 const handleEnterPress = (e) => {  
2   if (e.key === 'Enter' && activeSuggestionIndex >= 0) {  
3     const selectedSuggestion = filteredSuggestions[  
4       activeSuggestionIndex];  
5     navigate(selectedSuggestion.link);  
6   } else if (e.key === 'ArrowDown') {  
7     setActiveSuggestionIndex((prevIndex) =>  
8       prevIndex < filteredSuggestions.length - 1 ? prevIndex + 1 :  
9       prevIndex  
10    );  
11   } else if (e.key === 'ArrowUp') {  
12     setActiveSuggestionIndex((prevIndex) =>  
13       prevIndex > 0 ? prevIndex - 1 : prevIndex  
14    );  
15   }  
16 }
```

Sourcecode 18: Keyboard Navigation Implementation

Here is how it works: The `handleEnterPress` function detects `Enter`, `ArrowDown`, and `ArrowUp` key presses. The arrow keys allow users to navigate through search suggestions. When a user presses `Enter` on a highlighted suggestion, the user is redirected to the corresponding FAQ section. This improves accessibility by allowing keyboard users to interact seamlessly with the search bar.

Conclusion

The combination of `useMemo`, `useEffect` and keyboard navigation optimizations greatly improves the performance and usability of the FAQ page. These optimizations:

- Reduce unnecessary re-rendering.
- Provide a smooth and dynamic search experience.
- Improve accessibility for keyboard navigation users.

By using these strategies, the FAQ page provides a very responsive and efficient user experience.

5.10. Test

Testing in code embodies a systematic and empirical approach to verifying software behavior through controlled experiments. By employing methodologies such as unit, integration, and end-to-end testing, developers formulate hypotheses regarding how individual functions and components should operate under specific conditions and then rigorously validate

these assumptions through precise, reproducible test cases.

Testing is generally stratified into three primary categories ²⁶:

- **Unit Testing:** This involves the evaluation of individual components or functions in isolation. Unit tests serve as the first line of defense by verifying that each component behaves as expected under controlled conditions.
- **Integration Testing:** Beyond isolated units, integration testing assesses the interactions among components. This approach ensures that the data flows and inter-component communications adhere to the expected patterns, thereby mitigating integration issues.
- **End-to-End Testing:** This level of testing simulates real user interactions by validating the complete application workflow. End-to-end tests are instrumental in identifying issues that might only surface in a full-scale operational environment.

Our frontend evaluation was initially structured around the Testing Pyramid methodology ²⁷, which advocates that approximately 70 percent of testing should consist of unit tests, 20 percent of integration tests, and 10 percent of end-to-end tests. This approach was first implemented on the registration component—a relatively compact module featuring multiple interactive elements such as routing functionality and API requests. However, even for this small component, the associated test code expanded to nearly twice the volume of the component’s source code. Given that our goal was to create a prototype, this method proved overly time-intensive. Consequently, we adopted a modified strategy whereby the frontend is manually tested before each merge request, with reviewers required to validate the changes live before approval.

The primary libraries used for testing the registration component include:

- **Vitest** ²⁸: Serving as the testing framework, Vitest provides functions such as `describe`, `it`, `expect`, `beforeEach`, `afterEach`, and `vi` for creating structured test cases, assertions, and mock implementations. Vitest facilitates the orchestration of tests and manages the test lifecycle.
- **React Testing Library** ²⁹: This library is used for rendering React components in a virtual DOM environment. The `render` function mounts the component within the context of various providers (e.g., `BrowserRouter`, `AuthProvider`, `TitleContextProvider`, and `WorkspaceContextProvider`), thereby mimicking the real application environment. The `screen` object offers a suite of query methods (e.g., `findByTestId`) that replicate how users interact with the application. The `cleanup`

²⁶<https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html>

²⁷<https://martinfowler.com/articles/practical-test-pyramid.html>

²⁸<https://vitest.dev/guide/>

²⁹<https://testing-library.com/docs/react-testing-library/intro/>

function is invoked after each test to ensure isolation between tests by clearing the DOM.

- **testing-library/user-event**: This utility is employed to simulate real user interactions, such as typing into input fields and clicking buttons. It provides a higher-level abstraction over direct DOM events, ensuring that tests are aligned with how end users would interact with the application.
- **React Router Dom Mocks** ³⁰: The test suite also includes mocks for routing functionality via react-router-dom. Custom mocks for useNavigate and useLocation are created using Vitest's mocking capabilities, enabling the simulation of navigation events and location states without invoking the actual router behavior.
- **Custom Hook Mocks**: The useAxiosRequest hook is mocked to simulate API requests without making real network calls. This controlled mock allows the test to verify that the correct parameters are passed and that subsequent navigation occurs as expected when a successful response is simulated.

5.11. Known Problems

In the concluding section of this chapter, the focus is directed towards identifying prevalent issues and software defects within the Frontend that have yet to be addressed owing to limitations in both technical and temporal capacities. The subsequent passages offer succinct elaboration on each of the concerns above.

- **ProjectCenter - Unrecognized Project Closure**

As elucidated in Section 5.3 (ProjectCenter), the functionality of opening and closing a project is as follows: during the period in which a user is engaged in a project, the project is rendered temporarily inaccessible to other users (boolean locked).

If the user exits the application without first closing the project via the designated button, for instance by closing the browser or the browser tab, the Backend does not receive a request and the locked status remains unchanged. Consequently, the project remains locked, even though the user is no longer actively working on it.

During the developmental process, an endeavor was made to identify the closure of the browser and/or the tab (event beforeUnload³¹) and to transmit a final request with the context project closing to the backend within the process. However, this approach proved unsuccessful, as it is not possible to distinguish between refreshing and closing a tab, and the user would close the project on refresh, thereby making the persistent login feature obsolete.

The issue could be addressed in conjunction with the backend through the implementation of a timer on the server, which re-releases the project after a specified duration without project-specific requests. This necessitates the user to re-enter the

³⁰<https://reactrouter.com/home>

³¹https://developer.mozilla.org/de/docs/Web/API/Window/beforeunload_event

project after the timer expires and the request is declined.

- **Editor - Mismatched Font-Sizes**

It has been determined that there is currently an error in the editor concerning the text size and its selection. The text size can be adjusted by selecting the text and then selecting the desired text size using the Combobox (see Section 5.5).

In instances where the cursor is positioned within text that has already been written, the combobox does not invariably adapt to the text size at the cursor's current position. It has also been observed that the text size displayed in the combo box does not correspond to the size in which the text is currently being written. The identified issues have not yet been resolved, and it remains unclear whether these issues can be resolved without replacing the editor library in use.

- **Editor - PDF export: Scaling Images**

An additional issue has been identified in the Editor in the context of images. The Editor facilitates the uploading of images and their integration into the document (see Section 5.5). In instances where images are of significant size, it has been observed that during the process of PDF export, these images are not displayed in their entirety, resulting in truncation at a specific point.

There are several approaches to solving this problem, some of which are highly dependent on the functionality of the editor library used (see Section 5.5). There are two possible solutions to this issue. On the one hand, it would be beneficial to ascertain whether it is feasible for the size of the image to be checked during the uploading and integration into the editor processes. In this case, the scaling could be automatically adjusted accordingly. Additionally, the implementation of a restriction on certain image sizes and resolutions, similar to the functionality of PDF uploads (see Section 5.4), could be advantageous. On the other hand, an alternative approach would be the integration of a manual adjustment functionality for the user, with the option of buttons or drag-and-drop, which would allow for greater customization.

- **Chat - Duplicate Prompt Recommendation Popup**

Another issue has been identified in the Chat functionality when interacting with the PDF Viewer. Specifically, when a PDF is uploaded and text is entered into the chat input field, a problematic behavior occurs if the user holds the left mouse button to mark text and inadvertently crosses the boundary into the PDF Viewer area (see Section 5.6). This action triggers a duplication of the prompt recommendation popup, which can occur multiple times, eventually obscuring the interface and rendering it unusable (see Figure 57).

The only known resolution at this time is to refresh the webpage, which is not an ideal user experience. The error arises because the React-PDF-Viewer³² component is monitoring text selections, and when the selection process starts outside the PDF and concludes within it, an `IndexSizeError` is generated due to the absence of a

³²<https://react-pdf-viewer.dev/>

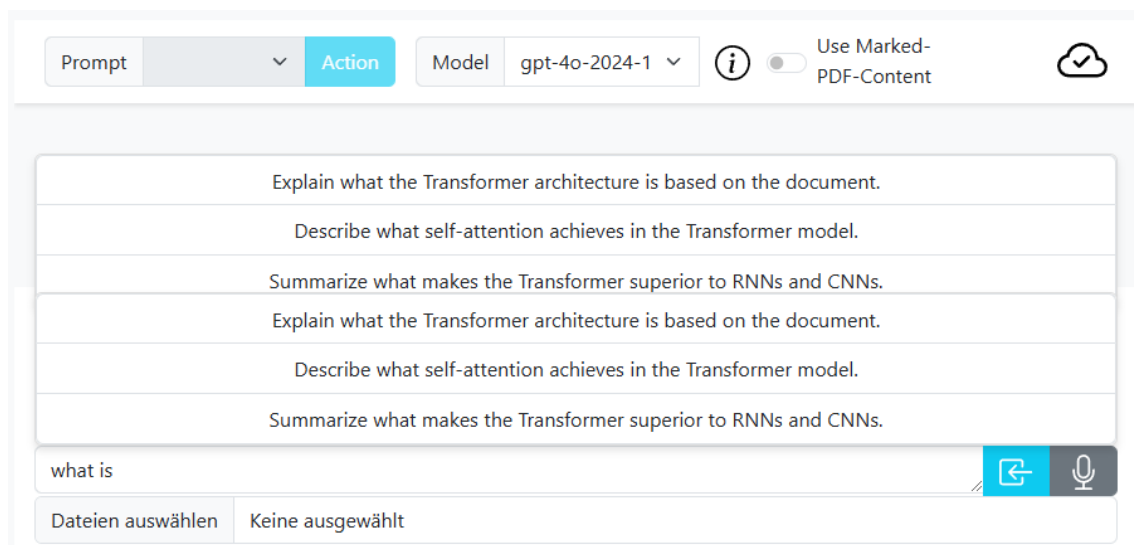


Figure 57: Duplication of the prompt suggestion

corresponding child element within the specified range. Figure 58 illustrates this bug: The left side 58 presents the standard display of the PDF, whereas the right side 58 demonstrates the effect of initiating a selection externally and terminating it within the PDF, which leads to the creation of empty pages. This error is further evidenced by the secondary navbar visible on the right side of the Figure 58, a feature absent in the normal display shown in Figure 58. Both Figures are from the demo of the highlight plugin of the React-PDF-Viewer³³.

To address this error, a React Error Boundary was implemented. React Error Boundaries are specialized components designed to intercept runtime errors within designated segments of a React component tree. They employ lifecycle methods — specifically, `getDerivedStateFromError` -, which updates the component's state upon error detection, and `componentDidCatch`, which logs the error details — to prevent the propagation of errors throughout the entire application. This mechanism is intended to enhance fault tolerance and ensure user interface stability by rendering fallback content when errors occur.³⁴

However, the underlying error, which originates from the external API (React-Pdf-Viewer), persisted despite the implementation of the error boundary. Given the complexity and resource demands associated with resolving this API-induced error, further remediation was deemed excessively time-consuming, if not potentially infeasible.

³³<https://react-pdf-viewer.dev/plugins/highlight/>

³⁴<https://react.dev/reference/react/Component#catching-rendering-errors-with-an-error-boundary>

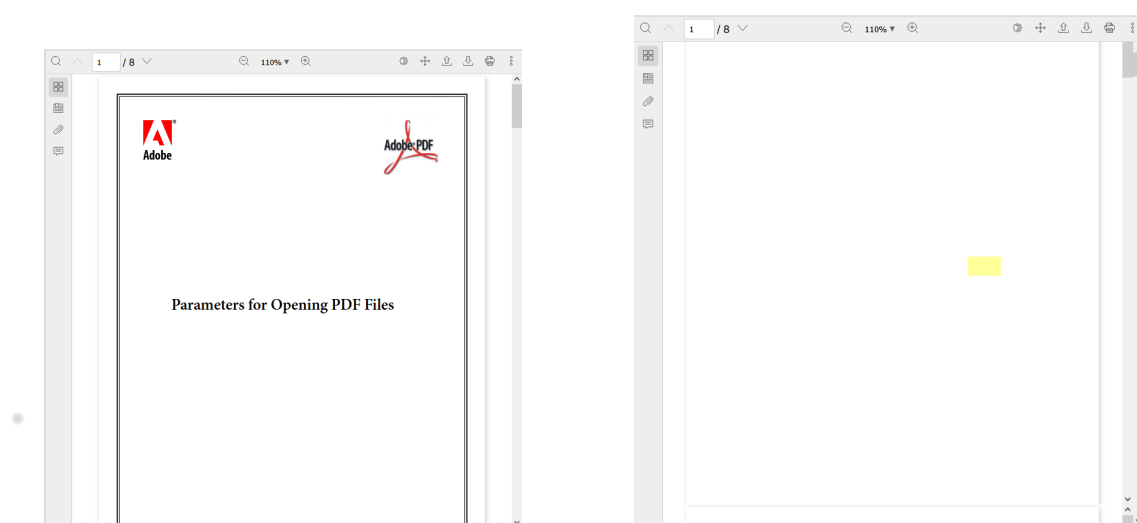


Figure 58: Comparison of normal(left) and error(right) display of the highlight plugin from React-Pdf-Viewer

6. Backend

This section covers the backend, from the overall structure of the project to its main components and configurations.

6.1. Overview of the Backend Specification

The project has been designed with one main entry point, the `app.js` file. It is responsible for loading the necessary configuration data, setting variables that will be used at runtime, collecting all subsequent endpoints into one file while defining the address at which they can be reached, and preparing the database with dummy data for testing purposes. All of these functions are described in more detail in the following chapters.

6.1.1. Configuration

The project contains two configuration files in total: `default.js` and `development.js`. The separate files represent different stages of development. The first file is used with default parameters, so if no other file is specified, the parameters within it will be used rather than specific ones. Development is used for the explicit purpose of working in a development environment, i.e. a local machine. Other configuration files could be ones such as deployment (mirroring a production environment) and production (being provided as a service on the World Wide Web).

The latter two files were not set up in the project as they were not needed, but they were created with the explicit expectation that further configuration files would be needed if the project was developed further and deployed using something like Docker. In the current project, the two existing configuration files are identical as they serve the same purpose - to be run on a local machine for development purposes.

6.1.2. Project Structure

As described in Section 6.1, the project has a main entry point. This entry point primarily imports and uses various controllers, which in turn depend on services, which in turn depend on repositories. Each of these classes has a specific subset of responsibilities that it will take on, representing a multi-tier architecture.

- **Controller:** These classes represent the service layer. They contain the explicit services/functionality provided by the backend in the form of endpoints that can be invoked. By default, they do not contain any logic other than receiving requests and extracting information from them, with a few exceptions described later in this section.
- **Service:** Classes of this type represent the business logic layer. They prepare and pre-process data according to rules and conventions agreed within the project. This layer as such specifically receives the raw data, creates new data, retrieves existing data using the repository classes and puts it into the correct form for operations directly on the database.

- **Repository:** The Repository class represents the data access layer. It does not modify any existing data and only performs CRUD operations on the database itself.

A deliberate violation of this layer structure was made in the case of a controller depending on services other than its own. This decision was made to avoid a service depending on another service, but it also resulted in some of the business logic being moved to the controller.

6.1.3. Dependencies

The node backend itself comes with multiple dependencies that help to provide its capabilities.

Albeit possible to implement all of the functionalities of it in plain node, the framework `express.js`³⁵ simplifies that process by providing an easy-to-use HTTP routing system as well as request and response handling. These are needed to enable the REST API endpoints so that the backend can be called from both, frontend and AI backend.

Another important dependency is the authentication middleware `passport`³⁶, which aims to secure the HTTP routes provided via a JSON Web Token. Without proper authentication the routes could be potentially accessed by third parties, which would constitute a security risk that needs to be accounted for. The entire process of encoding and decoding JWTs is described in 2.3.6. The required secret access and refresh tokens are part of the `default.js` config. To utilize JWTs for authentication purposes, the additional dependency `passport-jwt` is needed which also directly depends on the `jsonwebtoken` library. This can be deducted from its respective section in the dependency tree. The `jsonwebtoken` dependency here is marked as “deduped”, since the library is also part of the `package.json` and is thus already satisfied through the regular npm install. This is an automatic checkup provided by node to avoid the creation of duplicate dependencies.

```
+-- passport-jwt@4.0.1
| +-- jsonwebtoken@9.0.2 deduped
| '-- passport-strategy@1.0.0
```

An important dependency is `cors` (Cross-Origin Resource Sharing)³⁷. Natively, browsers enforce same-origin policy (SOP), which prevents JavaScript running on one origin from making requests to resources on another origin. To circumvent this problem, the server needs to explicitly allow cross-origin requests using CORS headers. The package in `Express.js` thus simplifies CORS by automatically setting appropriate headers.

The backend must be able to persistently store a multitude of relevant information, regarding user data, prompt data, chat histories and even the information contained in the

³⁵<https://expressjs.com/>

³⁶<https://www.passportjs.org/>

³⁷<https://www.npmjs.com/package/cors>

prompting frameworks. For that specific purpose, it was decided that a NoSQL database should be used, since relations between different tables were not required, given that the main storages – the user and the workspace storages – could be best done by nesting information rather than creating and referencing new tables. This, for instance, relates to the saved prompts for a user or the chat history for a workspace. It was thus chosen to use the NoSQL database MongoDB. Thus, a driver that allows the node application to connect to the MongoDB as well as to perform operations such as insert, update, delete or to query for data, is needed. In this case, the `mongodb`³⁸ dependency.

Like the frontend, the backend also uses the `axios`³⁹ package, which enables promise-based HTTP requests and handles the JSON response. While express provides the necessary routes and their functionality, axios calls APIs from different servers, in this case, for instance, the AI backend, where it is used as a promise-based part of the WebSocket connection between client and backend that handles the chat requests that need to be processed by the AI backend. Thus, express and axios together provide a complete functionality of handling incoming and outgoing requests. Down below is the dependency tree of axios.

```
+-- axios@1.7.5
| +-- follow-redirects@1.15.6
| +-- form-data@4.0.0
| | +-- asynckit@0.4.0
| | +-- combined-stream@1.0.8
| | | '-- delayed-stream@1.0.0
| | '-- mime-types@2.1.35
| | '-- mime-db@1.52.0
| '-- proxy-from-env@1.1.0
```

Since natively node is not capable of properly extracting the encoded data from a request via `req.body`, the dependency `body-parser`⁴⁰ is used to help extracting the request body from an HTTP request and convert it into a JS object. The `body-parser` is therefore a crucial element that is used in all endpoints that are required to process incoming request data. In the `app.js` the max file size that can be handled is specified.

Another noteworthy dependency is `swagger`⁴¹, which provides documentation and visualization of the implemented API endpoints. This allows for easier testing and removes the need for a manual endpoint documentation. In this case, the `swagger` options are also set to support authentication via JWT, thus ensuring secured endpoints. The `swagger` UI is accessible via `/api/swagger`.

³⁸<https://www.mongodb.com/docs/drivers/node/current/>

³⁹<https://axios-http.com/docs/intro>

⁴⁰<https://www.npmjs.com/package/body-parser>

⁴¹<https://swagger.io/>

6.2. Database Integration

This section will provide a overview of the database which is used for the storage of both dynamic and static data. The decision was taken to store static data, such as the available models, in the database rather than in a local file. This approach ensures that the data is available in persistent storage.

For the purposes of this project, the decision was made to use a MongoDB solution as one instance of a NoSQL database. The reasoning behind this decision was that the overhead of a relational database was deemed unnecessary within the project's scope, with the time saved from not using a relational model being invested in development time. The database contains a total of four collections: **Users**, **Model**, **PDF** and **Workspaces**. Each of these stores data for different aspects of the project.

6.2.1. Users

The **Users** collection is used to represent individual users of the service and stores data directly related to each individual instance - Figure 59 shows the structure of the collection along with its datatypes which can be broken down as follows:

- **_id**: This represents the default objectId used by MongoDB as a unique identifier for each individual collection.
- **userName**: The name of the individual user is represented as a string.
- **passwordHash**: The hashed version of the user's password is represented as a string.
- **salt**: The salt for this user's passwords is represented as a string.
- **role**: The individual user's role is stored as a string. Within the current project, the role of the user has no direct purpose, and the two existing roles are **Dummy** for Dummy Users and **User** for default users. The role serves as a differentiator.

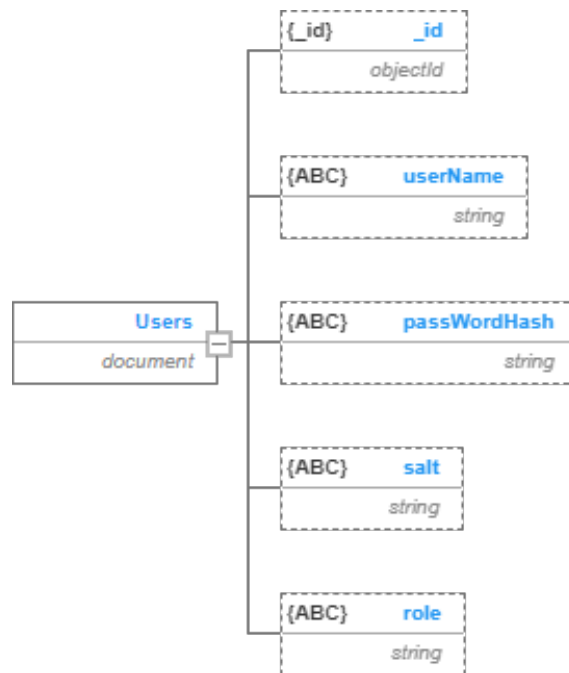


Figure 59: Schema representation of the user collection

6.2.2. Model

As illustrated in Figure 60, the `model` collection represents the individual models and their corresponding data. This collection is not intended to be modified at runtime and stores static data.

- `_id`: This represents the default `objectId` used by MongoDB as a unique identifier for each individual collection.
- `name`: The name of the model is stored as a string.
- `cost`: The per token cost of the model stored as a string.
- `type`: The model type is stored as a string. The existing types are: 'text' for multimodal models and models that can only work with text-based inputs; 'image' for models that can only work with image-based inputs; and 'audio' for models that can only work with audio-based inputs.
- `customId`: Each model is assigned a unique numeric ID. This is because not all data structures are capable of handling `objectIds`. As a result, a separate custom ID is stored to represent the model.

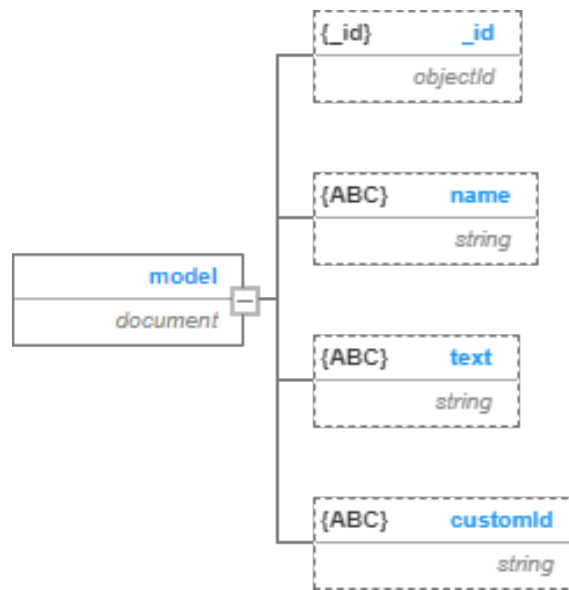


Figure 60: Schema representation of the model collection

6.2.3. PDF

The PDF collection represents the various prompting guidelines in a static manner, meaning the data is never altered during runtime and is simply displayed as it is. The collection's structure is outlined in Figure 61, and the individual fields are broken down as follows.

- **_id**: This represents the default objectId used by MongoDB as a unique identifier for each individual collection.
- **base64Content**: The PDF itself as a string in base64 format.
- **customId**: A numeric id assigned to the PDF, as not all data structures can handle an objectId.
- **name**: A string representation of the PDF name.
- **summary**: A summary of the key points of the PDF as a string.

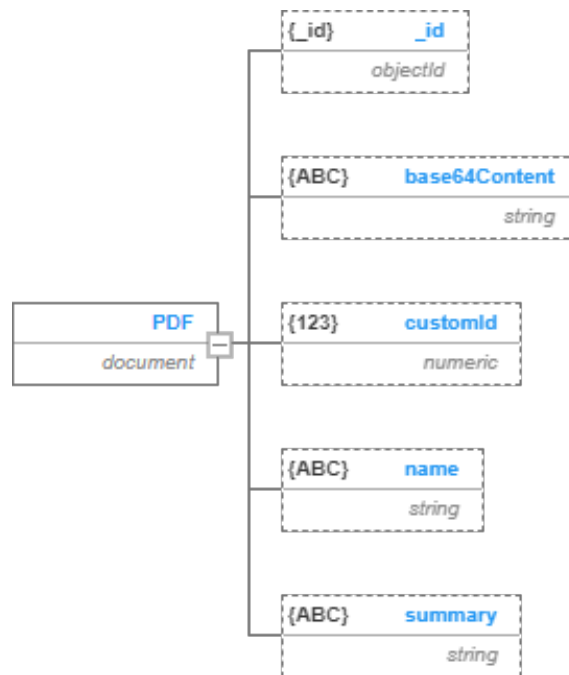


Figure 61: Schema representation of the PDF collection

6.2.4. Workspaces

Workspaces represent the totality of the data that the user has stored for the instance of the ProjectCenter as shown in Section 5.3. It is a large collection that represents several dynamic aspects of the Project Center; the individual fields are explained below.

- **_id**: This represents the default objectId used by MongoDB as a unique identifier for each individual collection.
- **id**: A numeric representation of an id, assigned by the backend on top of the objectId, as not all data structures can handle objectIds.
- **name**: The name of the workspace as a string.
- **description**: The description of the workspace as assigned by the user in the form of a string.
- **participants**: An array containing the participants of the workspace.
 - **id**: The objectId of the user who is a participant in the workspace.
 - **name**: The name of the user in the workspace as a string.
 - **isOwner**: A Boolean indicating whether the user is the owner of the workspace - it is used to check whether the user has permissions.

- **locked**: A boolean that indicates whether the workspace is currently in use or not, being true if the workspace is in use and false otherwise.
- **editorData**: An array representing the data stored for the editor (see Section 5.5).
 - **editorState**: A string representation of the contents of the editor. Non-string data is stored in the appropriate base64 encoded format as part of the string.
 - **lastSaved**: The date when the editorState was last saved as a timestamped string.
- **prompts**: An array of prompts from the PromptDesigner (see Section 5.8).
 - **id**: The id of the prompt as a string.
 - **name**: The name of the prompt as a string.
 - **description**: The description of the prompt as a string.
 - **prompt**: The prompt itself as a string.
 - **issues**: An array containing several issues.
 - **id**: A numeric id for the issue.
 - **fix**: A boolean representing whether the problem should be fixed or not.
 - **description**: The description of the problem as a string.
 - **improvement_suggestion**: The suggested improvements to fix the described issue as a string.
 - **rating**: The rating of the prompt on a scale from 0 to 10 as a string.
 - **temperature**: The temperature for the model that was selected by the user as a string.
 - **model**: The id of the selected model as a numeric value.
 - **framework**: The id of the selected framework as a numeric value.
 - **goal**: The goal of the prompt design process as defined by the user in the form of a string.
- **chatHistory**: The chat history of the workspace as a array.
 - **userId**: The objectId of the user who sent the message, represented as a string instead of an objectId. It is null if the message was sent by the server.
 - **chatMessage**: The message sent as a string by the user or the server.
 - **chatAudio**: The audio sent by the user as part of the message as a base64 encoded string.
 - **chatImage**: An array of images, each of which is a base64 encoded string.
 - **timeSent**: The date the message was sent, along with a timestamp as a string.
- **pdfs**: An array of PDFs used by users within the workspace.

- **id**: A numeric id for the PDFs.
 - **filename**: The name of the PDF as a string.
 - **content**: The PDF itself as a base64 encoded string.
 - **fileSize**: The size of the PDF, expressed in bytes as a numeric value.
 - **mimeType**: The mimeType of the PDF as a string.
 - **file_id**: The id of the PDF in the vector store as a string(see Paragraph 7.2.4).
 - **uploadDate**: The date at which the file was uploaded with a timestamp as a date.
- **vector_store_id**: The id of the vector store to which all files in the workspace have been uploaded as a string.
 - **assistant_id**: The ID of the OpenAI Assistant assigned to the workspace.

6.3. Authentication and users

This section provides an overview of the various components that are part of the authentication and registration process for users within the project. These two components have been combined into one chapter because they are closely related.

Figure 63 shows all the processes that directly involve the user, the figure will be broken down into individual steps in the future. Note that the figure is a simplified representation of the actual classes, methods that are not important for the purpose of explaining the functionality, such as helper methods, have been excluded in some cases for the sake of brevity, while others that are important for understanding the workflow have been left in. Not all data represented as classes is actually a class, sometimes it is just a collection of functions such as the password class. The base repository is not explicitly addressed, although it is shown in the diagram, as it is used by all databases and only implements basic CRUD operations on the database.

6.3.1. BearerHandler

The BearerHandler class has one notable purpose, which is to extract information from the bearer token for identification purposes. Its primary function within the project at the time of writing is to extract the user's ID from the encrypted bearer token so that the user can be uniquely identified using it, for which it provides the **getID** method.

getID: Returns the ID of a user that was encoded in the bearer token.

6.3.2. CipherHelper

The CipherHelper class is a collection of helper methods for encoding information, formatting information, and generating access and refresh tokens.

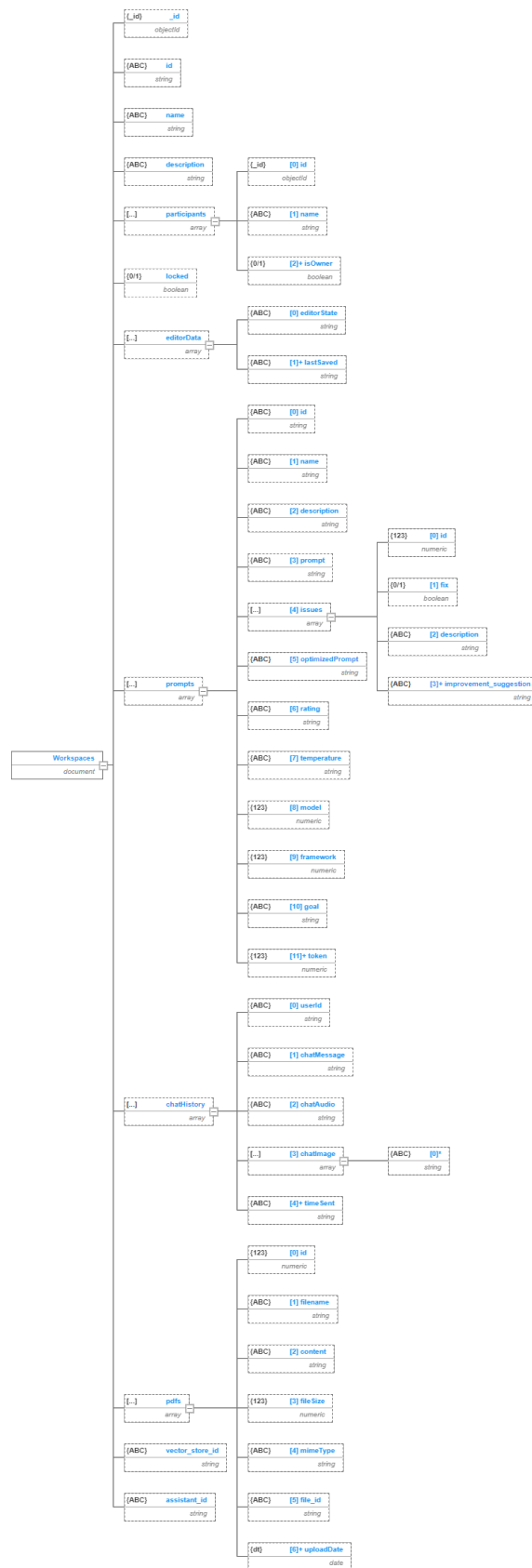


Figure 62: Schema representation of the workspaces collection

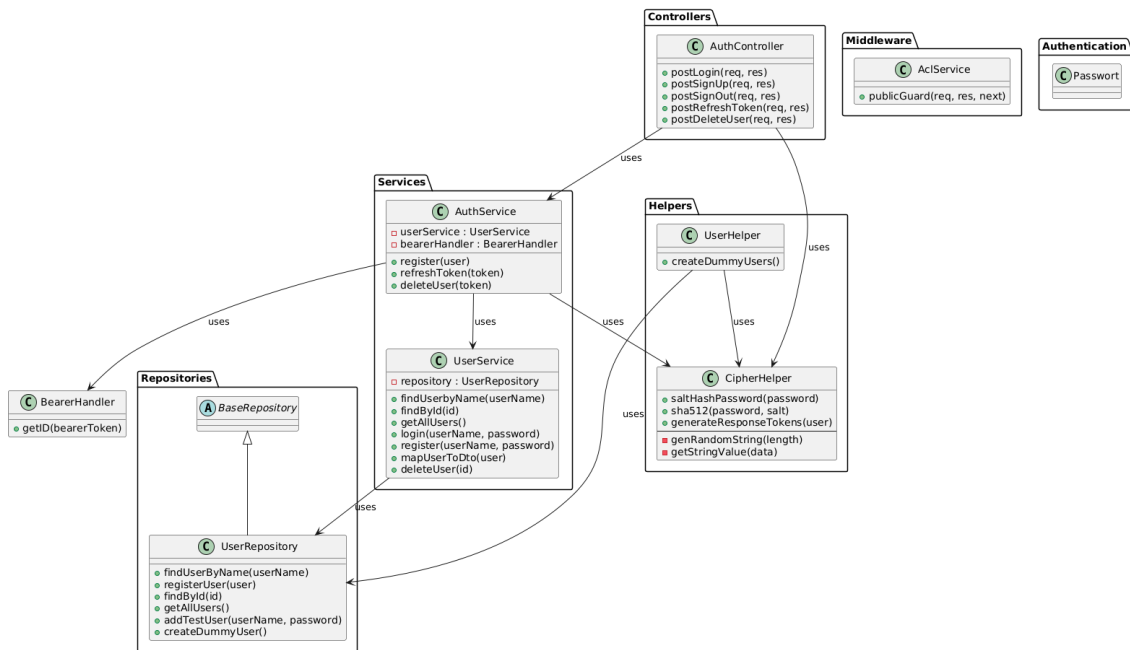


Figure 63: UML User

- **genRandomString**: Generates a random hexadecimal string of a specified length.
- **getStringValue**: Validates that the input is a string or buffer.
- **sha512**: Generates an SHA-512 hash for a given password and salt.
- **saltHashPassword**: Generates a salt and hashes the password with it.
- **generateResponseTokens**: Generates access and refresh tokens for a given user.

6.3.3. AuthController

The **AuthController** class is a collection of endpoints for the user to access any logic directly related to authenticating a user, such as registering a new account, deleting an existing account, logging in or logging out. The endpoints are not represented as `/login` but as a method using a combination of the HTTP request type and the method name, so a POST method running under the `/login` endpoint will be represented as **postLogin**.

- **postLogin**: Router for the user login, expects valid logging credentials as defined in the local passport strategy.
- **postSignup**: Router for user registration. Expects a user password and name.
- **postSignOut**: Route for user sign-out. This is a dummy endpoint for the time being with no direct effect.

- **postRefreshToken**: Route for refreshing a user's token. Expects the refresh token of the user.
- **postDeleteUser**: Route for deleting a user account. Expects the access token for the user.

6.3.4. AuthService

The AuthService provides the business logic corresponding to the AuthController. It handles user registration, deletion and token refreshes.

- **register**: Registers a new user by hashing their password and saving their information in the database.
- **refreshToken**: Refreshes the access and refresh tokens for a user if the provided token is valid.
- **deleteUser**: Deletes a user based on the provided token.

6.3.5. UserService

The UserService provides business logic similar to the AuthService in relation to the user. It mimics the functions already described in Section 6.3.4, which will not be repeated here, but also provides additional functionality.

- **findUserByName**: Retrieves a user by their username.
- **findById**: Retrieves a user by their unique ID and maps it to a DTO format.
- **getAllUsers**: Retrieves all users from the repository.
- **mapUserToDto**: Maps user data to a DTO format.

6.4. Backend Functionality

6.4.1. UserRepository

The UserRepository provides the data access layer for directly performing operations on the database based on the prepared data from the UserService.

- **findUserByName**: Retrieve a user by their username from the database.
- **registerUser**: Registers a new user in the database.
- **findById**: Retrieves a user by their MongoDB Object ID.
- **getAllUsers**: Retrieves all users from the database.
- **addTestUser**: Helper method to add a test user to the database.
- **createDummyUser**: Creates dummy users. This method is deprecated and no longer used, instead use the UserHelper's method is used.

6.4.2. PDF Viewer

The functionality for the PDF Viewer is provided on the backend in the following classes (see Figure 64).

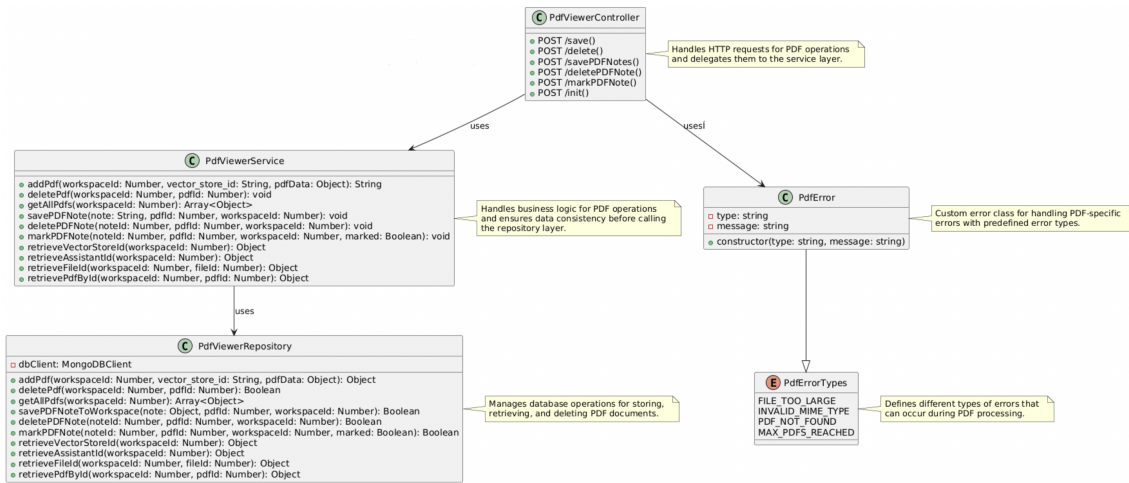


Figure 64: UML PDF Helper

The functions are organized in a multi-layer architecture consisting of three main components: **PdfViewerController**, **PdfViewerService** and **PdfViewerRepository**. The Application programming interface (API) to the frontend is in the controller layer, which provides various methods. Connected with the controller layer is the service layer, where the data is prepared and processed in. Once validated, the data can be written to the database in the repository layer or read as needed. This structure enables a clear separation of responsibilities, ensuring scalable code. Additionally, separating technologies simplifies future adaptations.

As mentioned above, the **PdfViewerController** provides several POST Endpoints to handle incoming HTTP requests. These methods can be accessed from the frontend and provide the following functionality:

- **POST /save:** Adds a new PDF file to a specified workspace after checking that the maximum number of PDFs has not been exceeded.
- **POST /delete:** Deletes a specific PDF file from a workspace based on its ID.
- **POST /savePDFNotes:** Stores a note for a specific PDF file in a workspace and assigns a unique ID to it.
- **POST /deletePDFNote:** Removes a specific note from a PDF file in a workspace.
- **POST /markPDFNote:** Updates the **marked** status of a particular note within a PDF file.

- **POST /init**: Initializes the PDF viewer by preparing all required data for use.

The service layer (`PdfViewerService`) implements the core business logic of the application. It performs tasks such as data validation, transformation and consistency assurance before passing information to the repository layer, which is responsible for database interactions. These two layers have completely different roles and should not be mixed. The service layer provides a clear separation between business logic and database structure, improving maintainability and extensibility.

In addition, the terms used to describe these layers should be introduced earlier for clarity, as otherwise the introduction section does not fully reflect their distinction.

- **addPdf**: Adds a new PDF file to a workspace and returns the ID of the added PDF file, provided the maximum number of PDF files has not been exceeded.
- **deletePdf**: Deletes a specific PDF file from a workspace, if it exists. Throws an error if the file is not found.
- **getAllPdfs**: Retrieves all PDFs of a specific workspace and returns them as an array of PDF objects.
- **savePDFNote**: Saves a note for a specific PDF file by giving it a unique ID and storing it in the database.
- **deletePDFNote**: Removes a specific note from a PDF file within a workspace.
- **markPDFNote**: Updates the `marked` status of a specific note within a PDF file.
- **retrieveVectorStored**: Retrieves the `vector_store_id` of a specific workspace and returns it as an object.
- **retrieveAssistantId**: Retrieves a specific workspace's `assistant_id` and returns it as object
- **retrieveFileId**: Searches a workspace for a specific PDF file by ID and returns the `file_id`.
- **retrievePdfById**: Retrieves a specific PDF file from a workspace using its ID and returns its full data.

The repository layer handles the direct communication with the database. It is responsible for CRUD operations (create, read, update, delete) and interacts with the MongoDB database via an instance of `MongoDBClient`. For this reason, this class also provides all the methods for interacting with the database.

- **dbClient**: An instance of the MongoDB client used to communicate with the database.

- **addPdf**: Creates a new PDF file in a workspace, gives it a unique ID, saves it to the database and returns the newly added PDF object.
- **deletePdf**: Removes a specific PDF file from a workspace and returns **true** if the file has been successfully removed and **false** otherwise.
- **getAllPdfs**: Retrieves all PDFs of a specific workspace from the database and returns them as an array of objects.
- **savePDFNoteToWorkspace**: Adds a new note to a PDF file, saves it in the mongo database and returns **true** if the operation was successful.
- **deletePDFNote**: Deletes a specific note from a PDF file within a workspace and returns **true** if the note was successfully removed.
- **markPDFNote**: Sets the **marked** status of a note in a PDF file to **true** or **false** and returns **true** if the operation was successful.
- **retrieveVectorStored**: Retrieves the **vector_store_id** of a given workspace from the database and returns it as an object.
- **retrieveAssistantId**: Retrieves the **assistant_id** of a specific workspace from the database and returns it as an object.
- **retrieveFileId**: Searches for a specific PDF file in a workspace and returns the **file.id** associated with it.
- **retrievePdfById**: Retrieves a specific PDF from a workspace by ID and returns its full data as object.

A special characteristic of this implementation is the use of `PDFError` and the `PDFErrorTypes` enum (see Figure 64). The class `PDFError` sets user-defined error codes for PDF-specific errors. This is where standardized error codes are defined that can occur when processing PDF files. These error codes are used by `PdfError` to generate consistent error messages. Instead of using generic errors, the system can provide specific error codes and messages using `PdfErrorTypes` and `PdfError`. This structure ensures clear error handling and better maintainability of the code:

- **FILE_TOO_LARGE**: File is too large.
- **INVALID_MIME_TYPE**: MIME type is not supported.
- **PDF_NOT_FOUND**: The requested PDF file does not exist.
- **MAX_PDFS_REACHED**: Maximum number of PDFs reached.

6.4.3. DocumentEditor

HTTP requests related to the editor are managed and processed in the Node.js backend using a multi-layer architecture. It is implemented using the three main components: EditorController, EditorService and EditorRepository, which form a logical chain from application logic to data storage. A schematic representation of this architecture is shown in Figure 65.

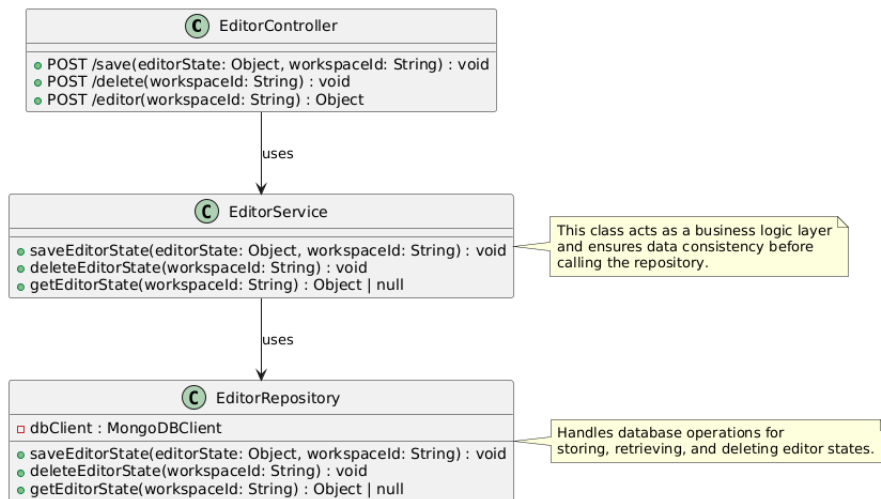


Figure 65: UML Editor

The EditorController handles HTTP requests and passes them to the service layer, ensuring a clear separation of concerns. It includes several POST methods, as shown in Figure 65.

- **POST /save:** Saves the current state of the editor for a given `workspaceId`.
- **POST /delete:** Deletes the saved editor state for the specified `workspaceId`.
- **POST /editor:** Get the current state of the editor for a given `workspaceId`.

The service layer handles the business logic of the application, including data validation, transformation and processing. The editor service provides methods for saving, deleting and retrieving editor states. It ensures that data is consistent and complete before it is passed to the repository layer.

- **saveEditorState:** Validates and saves the transferred editor state.
- **deleteEditorState:** Deletes the editor status for the specified `workspaceId`.
- **getEditorState:** Gets the current state of an editor or returns `null` if no entry exists.

The repository layer separates business logic from database interactions, making it easier to switch databases without affecting other parts of the system.

It handles all CRUD operations, with the editor repository using MongoDB Client to store and retrieve editor states. This design improves maintainability and customization.

- **saveEditorState:** Persists the editor status in the database depending on the workspace.
- **deleteEditorState:** Deletes the editor data of the workspace from the database.
- **getEditorState:** Retrieves a saved editor status.

The repository layer ensures strict separation between database interaction and business logic. This modular approach allows changes in database technology to be made without requiring changes to higher-level components.

The layered architecture provides a structured separation of concerns that improves code maintainability. Rather than using vague terms such as **extensible** or **scalable**, the advantage lies in the ability to modify and optimize each layer independently. By decoupling the business logic from the database, the backend remains adaptable to new requirements or evolving technologies, while maintaining clarity and modularity.

6.4.4. Chat

The Chat is one of TextVision's key features as it allows users to directly communicate with a selected AI model, thus assisting them with their specific tasks. To achieve this, a connection from the chat UI element to the backend is needed in order to effectively process incoming and outgoing messages. Also, a connection to the AI backend is needed where the AI models are called, and their responses are handled which are then sent back to the backend and ultimately the user. Although the chat system underlies the same multi-layered pattern consisting of controller, service and repository, in this case, REST API endpoints are not the only form of relevant routes present in the controller class.

Since a chat is supposed to consist of multiple messages between client and server, the use of mere REST API endpoints for the means of communication comes with certain drawbacks compared to some other protocols. For a chat system, it would be more advisable to follow a protocol that is bi-directional, as opposed to a request-response-based approach like REST, as in a bi-directional approach, both, server and client may send messages anytime, thus simplifying client-server communication and reducing time (Skvorc et al., 2014, p. 2). One of those approaches is the so-called WebSocket protocol. TextVision's chat function is thus based on a WebSocket connection that is reachable via the `/chat` route. Instead of handling different functionalities with different routes, the nature of WebSocket suggests that only one WebSocket connection is established that then handles different use-cases, differentiating by other means. In this case, client messages that arrive at the backend via the WebSocket are in JSON format and include a key **action** from which the actual functionality is derived. Inside the WebSocket function the different actions are handled with a switch-case statement. The following valid actions exist:

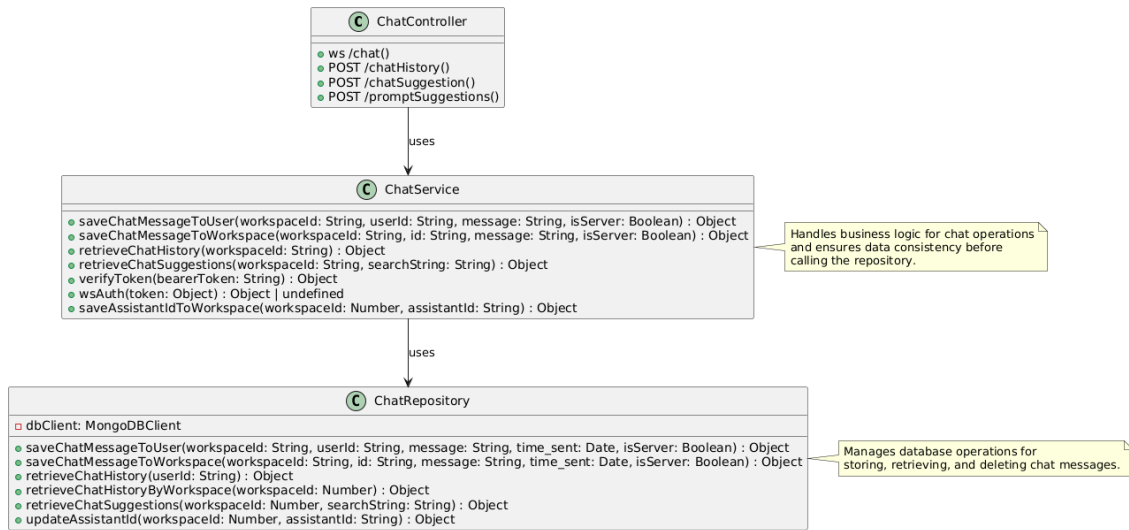


Figure 66: UML Chat

- **POST auth:** This case is for authorization only and authenticates users for the WebSocket by the authentication token that is also extracted from the JSON message.
- **POST create:** The create case adds a new group to the connection. A group functions as a collection of clients that are all connected to the same WebSocket, thus being able to commonly utilize it. The corresponding asynchronous function of the controller class is `createGroup(groupName: String, password: String)`.
- **POST join:** Via this case an authenticated user can join an existing group. The corresponding asynchronous function of the controller class is `POST addToGroup:..`
- **leave:** Via this case an authenticated user can leave an existing group. The corresponding asynchronous function of the controller class is `POST removeFromGroup`.
- **chat:** The most important case. Here, messages sent from the client via the chat system are handled. Incoming messages are first saved to the database using the ChatService and ultimately the ChatRepository and are then forwarded to the AI backend via a POST request to the /chatRequest route. The response message from the AI backend is then also saved to the database. Ultimately, the response message is also sent back to the client via WebSocket so that it is visible in the actual chat UI.

Similar to the other controller classes, the ChatController, next to the WebSocket, also contains several REST API endpoints that are callable via POST request. These endpoints are:

- **POST /chatHistory:** Via this route the entire chat history that is associated with the provided workspace ID can be retrieved.

- **POST /chatSuggestion:** This endpoint returns possible suggestions for chat messages.
- **POST /promptSuggestion:** This endpoint returns possible suggestions for prompts, based on a given unfinished prompt, model and temperature. The suggestions are also based on the PDFs and the chat history associated with the respective workspace. The latest suggestions are formed in the AI backend that is reached via route /promptSuggestions and are then sent to the backend within the response of the POST request.

Like the other functionalities, the ChatService contains the business logic, thus separating it from the API endpoints in the controller and the database calls in the repository. Notable functions of the ChatService are:

- **saveChatMessageToWorkspace:** This method creates a timestamp of the very time it is called and then forwards it along with the chat message that is to be saved in the database to the ChatRepository.
- **retrieveChatHistory:** This method calls the respective repository method to retrieve the entire chat history of a workspace that is specified via ID.
- **retrieveChatSuggestions:** This method calls the respective repository method to retrieve the chat suggestions of a workspace that is specified via ID.
- **saveAssistantIdToWorkspace:** This method saves or updates the assistant ID of the workspace that is specified via workspace ID.

The ChatService also contains methods to verify a bearer token and to authenticate to the WebSocket.

The repository layer which inherits from the BaseRepository then handles all database calls by using the instantiated MongoDB client. The ChatRepository includes the following methods:

- **saveChatMessageToWorkspace:** Saves a new chat message to the chat history associated with the workspace that is derived from the provided workspace ID.
- **retrieveChatHistoryByWorkspace:** Database call that retrieves the entire chat history associated with the provided workspace.
- **updateAssistantId:** Saves or updates the assistant ID of an existing workspace that is specified via ID.

6.4.5. PromptDesigner

All functionality related to the PromptDesigner is encapsulated in the four related classes (see Figure 67). The classes are the primary interface for managing the prompts in the node.js backend and handle all incoming requests.

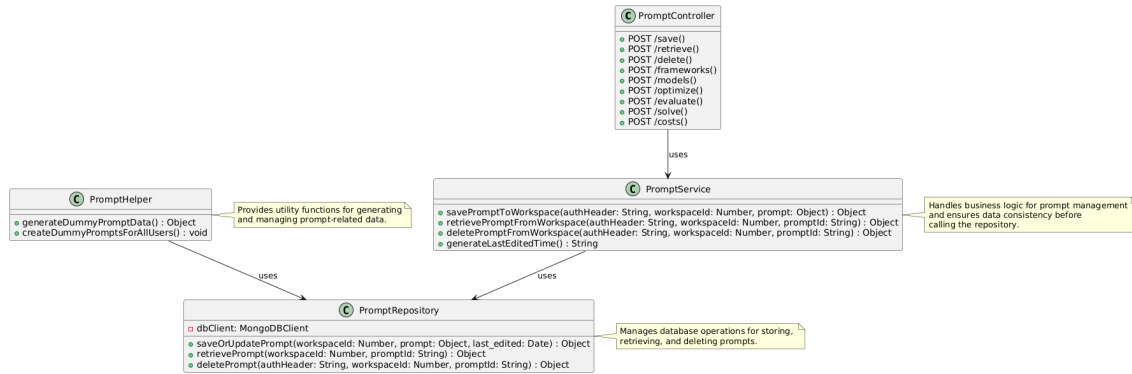


Figure 67: Prompt API

In the first step, the `PromptController` is the interface between the backend and the frontend. It intercepts the incoming HTTP requests. The `PromptController` provides the following functionality:

- `POST /save`: Saves a new prompt for a particular `workspaceId`
- `POST /retrieve`: Retrieves a saved prompt using the `workspaceId` and `promptId`.
- `POST /delete`: Deletes a saved prompt with the `workspaceId` and `promptId`.
- `POST /frameworks`: Retrieves the frameworks for prompts.
- `POST /models`: Lists available models for optimization of prompts.
- `POST /optimize`: Performs an optimization of a given prompt and return issues.
- `POST /evaluate`: Evaluates a prompt based on the selected framework.
- `POST /solve`: Solves all selected issues.
- `POST /costs`: Calculates the cost of a specific request.

The controller layer works with the service layer, which handles business logic. It processes prompt data and provides methods for saving, deleting, and retrieving prompts. Before passing data to the service layer, the controller layer ensures its consistency and completeness.

- `savePromptToWorkspace`: Saves a prompt for a given `workspaceId`.
- `retrievePromptFromWorkspace`: Retrieves the saved prompt using the `workspaceId` and `promptId`.
- `deletePromptFromWorkspace`: Deletes a saved prompt.

- `generateLastEditedTime`: Generates a timestamp for the last processing of a prompt.

The repository-layer manages the database operations for storing, retrieving and deleting prompts. It encapsulates the direct interaction with the database in a class. This separates the business logic from the database interaction.

- `saveOrUpdatePrompt`: Saves or updates a prompt in the database.
- `retrievePrompt`: Retrieves a saved prompt from the database.
- `deletePrompt`: Deletes a saved prompt from the database.

A special feature of prompt management is the use of a helper class. This class provides functions for creating and managing dummy prompts. This class is primarily used for testing and development purposes to automatically create prompts for user with the `Dummy` role and store them in the dummy database.

- `generateDummyPromptData`: Creates a random test prompt data structure.
- `createDummyPromptsForAllUsers`: Generates dummy prompts for all users and saves them via `PromptRepository`.

Essentially, `generateDummyPromptData` is a utility function designed to create a placeholder or test object containing typical prompt fields and their sample values for use in development or testing scenarios:

```
1 const generateDummyPromptData = () => {  
2   return {  
3     name: "Sample Prompt", // Name of the prompt  
4     description: "A brief description of the prompt", //  
       Description of the prompt's purpose  
5     prompt: "Summarize this topic", // The main content or  
       question for the prompt  
6     model: "1", // Model ID used for this prompt  
7     framework: "3", // Framework ID associated with this  
       prompt  
8     temperature: "0.2", // Temperature setting for AI  
       processing  
9     cost: "0.64", // Estimated cost for running the prompt  
10    token: "1345", // Token count for the prompt  
11    rating: "4", // Initial rating for the prompt  
12    issues: [ // List of issues related to the prompt  
13      { id: "1", fix: false, issue: "Example issue 1" },  
14      { id: "2", fix: false, issue: "Example issue 2" }  
15    ],  
  }
```

```

16     optimizedPrompt: "This is an optimized example prompt" //
        Optimized version of the prompt
17   }
18 };

```

6.4.6. Workspaces

As TextVision was conceptualized with the goal of providing a user-friendly application to collaboratively work with PDFs, a solution of having multiple users in real-time work on the same documents was needed. This would in turn require the participating users to be logged in and be part of a dedicated environment that, at a bare minimum, can be joined and left. Also, since those environments and their contained work are required to be saved and restored on demand, the entire related data needs to be persisted in a database. The described feature was later named “workspaces” to hint at its collaborative nature.

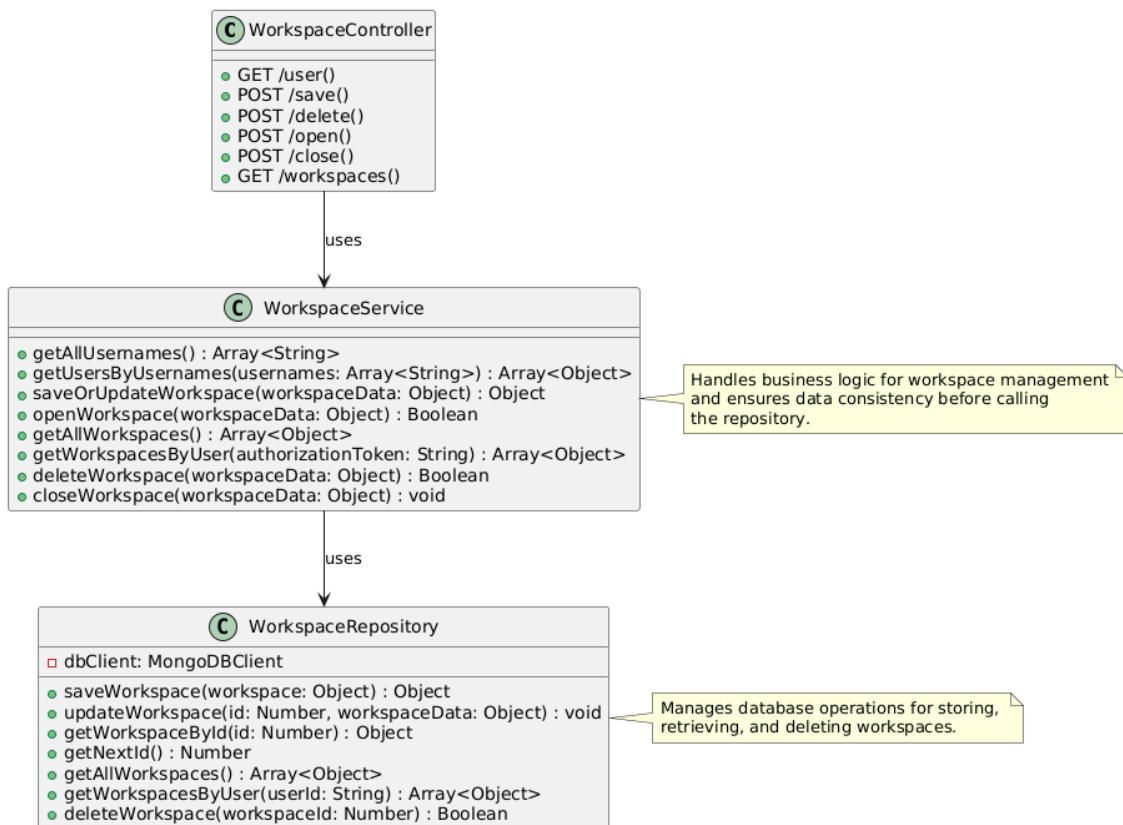


Figure 68: Workspace API

Like the other functionalities in the backend, the workspaces underly a multi-layered pattern consisting of controller, service and repository layer. Analogously, the controller layer contains the REST API endpoints, the service layer manages the business logic

while the repository layer is solely responsible for calling the database. The controller layer contains the following API endpoints that are callable with POST requests via route `/textVision/workspace`:

- **POST /user**: Endpoint to return all users.
- **POST /save**: Endpoint to save a new workspace, requiring a workspace ID, a name, description, and its participants in the request.
- **POST /delete**: Endpoint to permanently delete an already existing workspace by ID.
- **POST /open**: Endpoint to open a previously locked workspace by ID.
- **POST /close**: Endpoint to lock a previously open workspace by ID. A locked workspace cannot be altered until it is opened again.
- **POST /workspaces**: Endpoint that returns all workspaces that are assigned to a user.

While calling the endpoints ensures that the appropriate response follows its respective request, the actual business logic is handled inside the service layer. The controller class instantiates an object of the `WorkspaceService`, so that its asynchronous methods can be called when needed. The service, on the other hand, instantiates an object of the `WorkspaceRepository`, calling upon it whenever its methods for database calls are required. The `WorkspaceService` consists of the following methods:

- **getAllUsernames**: Returns all user by utilizing a `userService` method.
- **getUsersByUsernames**: Returns all users by their specific usernames by utilizing a `userService` method.
- **saveOrUpdateWorkspace**: Saves a new or updates an already existing workspace with the provided data. The latter operation only works if the existing workspace is not currently locked.
- **openWorkspace**: Opens an existing and previously locked workspace. The user must be authorized to carry out this operation.
- **closeWorkspace**: Similarly, locks an existing and previously open workspace. The user must be authorized to carry out this operation.
- **getWorkspacesByUser**: Returns all the workspaces associated with the user ID extracted from the authorization token.
- **deleteWorkspace**: Permanently deletes an already existing workspace from the database. The workspace may not be currently locked when this operation is performed.

The `WorkspaceRepository` which inherits from the `BaseRepository` instantiating the MongoDB client then specifies the appropriate database collection and implements the following asynchronous methods to save, update, delete and retrieve workspaces and its related data from the database:

- **saveWorkspace:** Saves a new workspace to the MongoDB collection “Workspaces”.
- **updateWorkspace:** Updates an already existing workspace in the database by ID.
- **getWorkspaceById:** Calls the database to retrieve the workspace associated with the provided ID.
- **getNextId:** Returns the next available valid integer ID by incrementing the currently biggest valid ID so that it can be associated with a new workspace.
- **getAllWorkspaces:** Returns all workspaces that are saved in the “Workspaces” collection.
- **getWorkspacesByUser:** Returns all saved workspaces associated with the given user ID.
- **deleteWorkspace:** Deletes an existing workspace from the database by ID.

6.4.7. Tests

In order to ensure that the created code works as intended and to account for possible bugs or edge cases, code testing is a reliable option that aims to potentially increase stability and security. Different types of testing vary from testing the mere functionality, like unit or integration tests to non-functional testing like performance and security testing. In general, it should be never assume that the created code is entirely correct, even after it was thoroughly tested, as testing might show the presence of errors, but it cannot prove their absence (Olan, 2003, p. 1).

The provided JavaScript classes (`promptHelper.js` and `PDFHelper.js`) are tested primarily through integration tests, emphasizing core functionalities and interactions with external components.

For example in the `promptHelper.js`:

- Tests verify that dummy prompts are correctly generated and added to workspace documents without existing prompts.
- Database interactions are validated by ensuring workspaces are retrieved accurately and prompts are successfully saved or updated.
- Error handling is tested for scenarios such as database connection failures or issues during prompt creation and storage.

Furthermore the `PDFHelper.js` is testing:

- Testing includes reading PDF files from a directory, accurately converting file contents into base64 strings, and associating these with the correct summaries.
- Ensures handling and logging of cases where summaries are missing or file reading issues occur.
- Validates successful integration with the `PDFService` for saving PDF data into the database.

Tests emphasize proper functionality, accurate data handling, and comprehensive error management.

6.5. Known Problems

This chapter will provide an overview of the issues and bugs that are known but could not be addressed within the scope of this project. The subsequent analysis will list issues belonging to three categories: firstly, unintended behavior; secondly, problems with the code regarding factors such as quality or expandability; and thirdly, security issues.

- In the event of a workspace being locked (i.e. when it is not accessible to other users), it is unable to exit this state and remains in it permanently. This issue can be temporarily resolved by ensuring that the workspace is never placed in this state, which, however, results in another problem, namely the potential for the workspace to be accessed by multiple users simultaneously, which can lead to synchronization errors between them.
- The Swagger documentation has not been maintained throughout the project, as it has been found to be more of an overhead than was initially anticipated. While this document serves as an alternative to the Swagger documentation, it is still less concise than a set standard.
- When performing operations on workspaces, the backend only verifies the existence of the user and the validity of their authentication; it does not verify whether the user is part of the workspace. This only affects operations on workspaces that do not require the user to be the workspace owner. As such, any registered user with a valid session could retrieve the workspace data of another workspace that they are not part of. This security issue is a remnant from the pre-workspace era when the user object stored workspace-related data.
- It has come to my attention that there are several instances of duplicated code within the project. One concrete example of this is the endpoint `deleteUser`, which checks if the validation header format is valid. This is supposed to be handled by the `bearerHandler` class as described in 6.3.1.
- The `assistant_id` 6.2.4 is not utilised within the project, due to an oversight during implementation and testing. The ID was not being passed correctly to the

ai-backend. However providing the `vector_store_id` 6.2.4 suffices for mapping it to the corresponding assistant.

- The way in which the various unique identifiers are assigned within the database is not consistent and also inefficient. This is because it iterates through all existing IDs until it finds the first unused ID.
- The database structure is inconsistent in terms of data types and naming conventions. Different cases are used, such as snake case and camel case. Variables representing the same thing for different objects have different names, such as `customId` and `id`. Numeric values are sometimes represented as strings, and in one instance, an `objectId` was saved as a string 6.2.4.
- The database is not able to guarantee integrity by default. This is not necessarily an error, but rather an inherent characteristic of MongoDB. Specifically, if a user is deleted, they will not be automatically removed from a workspace, which compromises integrity. The necessary logic to guarantee integrity was intentionally ignored within this project.

7. AI-Backend

The AI backend features a modular architecture optimized for processing text and image data through selected AI models. It serves as the core processing unit of the TextVision application, managing AI model integration, data transformation, and task execution. While the node backend itself processes input data, its primary role is to coordinate AI model interactions, ensure format compatibility, and orchestrate processing pipelines via standardized REST API endpoints.

The implementation follows software engineering principles by separating handlers, processors, and utility modules. Abstraction layers ensure standardized AI model integration while maintaining consistent interfaces. This structure enables dynamic model selection and parallel processing to efficiently handle computationally intensive tasks.

The system includes a suggestion mechanism that predicts user intentions based on interaction patterns and historical data to enhance efficiency. Additionally, it implements a prompt evaluation and optimization system that assesses prompt quality according to defined linguistic and contextual criteria, improving the utilization of language model capabilities. The AI backend also incorporates vector-based retrieval-augmented generation (RAG) for document analysis, allowing it to process large document collections by segmenting content into smaller chunks and referencing information across document boundaries.

The subsequent sections detail the system requirements, architectural design, implementation specifics, and key components comprising the AI-Backend system.

7.1. Description and Overview

This section provides an overview of the AI-powered backend system. The system is designed to process and analyze text, image, and audio data using multiple AI models, including Whisper for speech-to-text conversion. Its modular architecture allows for the integration of additional models and data formats, enabling adaptability to evolving requirements.

Key architectural highlights include:

- Modular design with clear separation of concerns
- Extensible AI model integration framework
- Robust file processing pipeline
- Advanced prompt management system
- Vector store-based retrieval augmented generation (RAG)
- Comprehensive security implementation

7.1.1. System Requirements and Development Stack

Development Environment Development tools and environments used:

- **IDE:** JetBrains PyCharm
 - Built-in virtual environment management
 - Git integration
 - Python debugging capabilities
- **Python Setup:** Python 3.8+ with pip and venv
- **Testing Tools:**
 - Pytest for unit and integration testing

System Prerequisites

Essential requirements for system deployment:

- **Hardware:** 4GB RAM, sufficient storage for document processing
- **Network:** Stable internet connection, Port 3003 available
- **API Access:** Valid OpenAI API key, HTTP/HTTPS support

Core Dependencies

Key libraries and frameworks specified in `requirements.txt`:

```
1 # Web Framework and API
2 fastapi>=0.112.0,<0.113.0      # REST API framework
3 uvicorn==0.30.5                # ASGI server
4 pydantic>=2.8.2                # Data validation
5
6 # AI and Processing
7 openai==1.57.2                 # OpenAI API integration
8 pymupdf4llm==0.0.17            # PDF processing
9 numpy>=1.21.0                  # Numerical computations
10
11 # Development and Testing
12 pytest==8.3.4                  # Testing framework
13 python-dotenv>=1.0.0           # Environment management
14 requests==2.32.3               # HTTP client
```

Configuration

System configuration requires:

- Environment variables in `.env`:

```
1 PGAIM_AI_API_KEY=your_api_key
```

- HTTP authentication setting in `config.ini`

Technology Stack

The system integrates four main technology components:

1. **Web Framework:** FastAPI is used to implement the REST API, providing asynchronous request handling and automatic API documentation.
2. **AI Integration:** Custom abstraction layer supporting:
 - Multiple AI models (7.2.1)
 - Vector store integration
 - Processing of text, images, and audio data
3. **Security Layer:**
 - HTTP Basic Authentication
 - Environment-based configuration
 - Secure file handling

7.1.2. Folder Structure

The AI Backend follows an organized modular structure that promotes code maintainability, scalability, and clear separation of concerns. Below is an overview of the corresponding folder and file structure.

Project Structure Overview

```
pgaim-ai-backend/
|-- .venv/           # Virtual Environment
|-- ki_api/          # Main Application Directory
|   |-- common/      # Shared Components
|   |-- docs/        # Documentation Files
|   |-- handlers/    # Handler Modules
|   |-- processors/  # Data Processing Modules
|   |-- tests/       # Test Suite
|-- .gitignore       # Git Ignore Configuration
```

```
|-- .gitlab-ci.yml      # GitLab CI/CD Configuration
|-- readME.md          # Project Documentation
|-- requirements.txt    # Python Dependencies
```

Organizational Principles

The structure follows these key principles:

- Modular organization for easy maintenance
- Clear separation of concerns
- Centralized configuration management
- Isolated test environment
- Scalable architecture design

This organization ensures efficient development, testing, and maintenance of the AI back-end system while maintaining a clear and logical structure.

7.2. Implementation

The AI Backend implements an architecture that integrates various AI models, prompt management, and document processing capabilities. Figure 69 illustrates the system’s modular architecture, highlighting the interactions between key components such as the `AIHandler`, `FileHandler`, and `PromptHandler`, as well as their connections to external storage and model APIs.

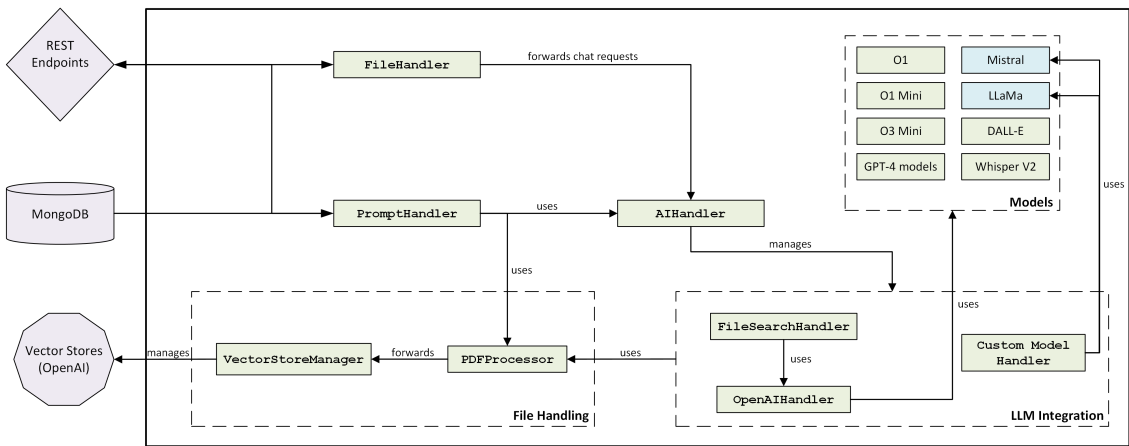


Figure 69: Architecture of the TextVision Python AI backend

The system architecture is structured around three fundamental processing subsystems, as depicted in Figure 69, which together provide comprehensive AI functionality.

The first subsystem, AI Model Integration, serves as the foundation for incorporating various LLMs. The **AIHandler** manages interactions with both API-based OpenAI models, including the GPT series, DALL-E, and Whisper, as well as self-hosted solutions such as Mistral and LLaMA. Model integration is achieved via the **OpenAIHandler** and **Custom Model Handler**, ensuring a unified interface for different AI providers.

Model flexibility is achieved through an abstraction layer, allowing dynamic switching between different AI models without modifying the core application logic. This is facilitated by a standardized system prompt structure and uniform output formatting, ensuring that responses from different models adhere to a consistent schema. As a result, applications interacting with the AI backend can rely on predictable response formats and function calls, regardless of the underlying model.

The second core subsystem, Prompt Management, plays a central role in the system's ability to interact effectively with AI models. The **PromptHandler**, as shown in Figure 69, retrieves stored prompts from MongoDB and refines their structure based on predefined criteria, improving response relevance and coherence. Additionally, the **FileHandler** forwards chat requests to the **AIHandler**, ensuring smooth communication between users and AI models.

The third major subsystem, Document Processing and RAG (Retrieval-Augmented Generation), provides advanced capabilities for handling and analyzing documents. This subsystem integrates the **PDFProcessor** for text extraction and preprocessing, while the **VectorStoreManager** manages vector embeddings for document retrieval. As depicted in Figure 69, the system utilizes OpenAI's vector stores to facilitate efficient retrieval operations. The **FileSearchHandler** further enhances document access by integrating AI-powered search capabilities.

These subsystems are orchestrated within a centralized Node.js backend, which acts as the primary coordination layer. As illustrated in Figure 69, this backend manages interactions between different components, including model selection and switching, document processing workflows, and user interface interactions. The system supports both API-based models and self-hosted solutions, ensuring compatibility with various deployment setups.

The following sections provide a detailed examination of each component's implementation, including class structures, methods, and functionalities, demonstrating how these elements work together to form a comprehensive AI processing pipeline.

7.2.1. Handlers

The handler implementation forms the core of the AI backend and manages the integration of different AI models as well as the processing of requests. The architecture is based on an abstract interface design that ensures uniform interaction with different AI backends.

AI Handler

The AI Handler provides a structured approach for integrating different AI models through an abstract interface design, as illustrated in Figure 70. This architecture follows the

Strategy pattern, allowing for dynamic switching between AI backends while maintaining a standardized interface.

The architecture consists of three main components:

- **AbstractAIBackend**: Interface for AI backend implementations
- **AIHandler**: Central coordination class
- **OpenAIHandler**: Concrete implementation for OpenAI models

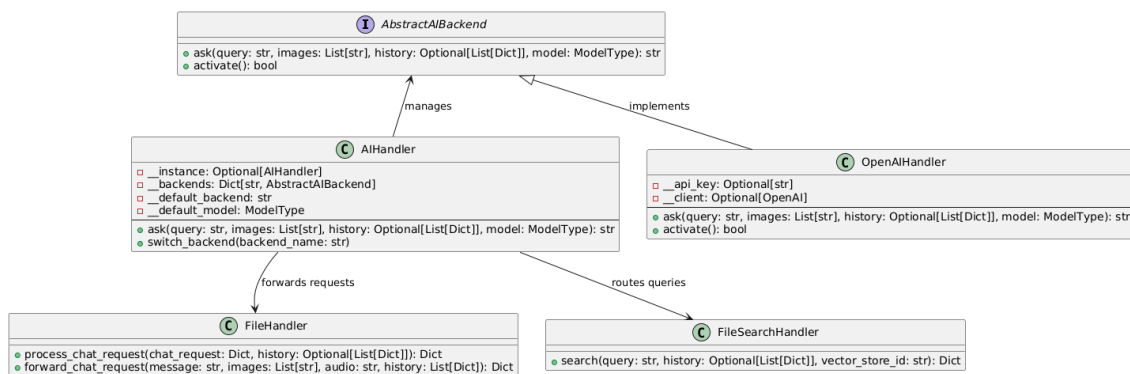


Figure 70: Handler Implementation

The **AbstractAIBackend** interface serves as the foundation for all AI backend implementations. It defines the contract that all concrete backends must fulfill, including:

- Vector store management capabilities
- Query processing through the `ask` method
- Backend activation and initialization
- Prompt evaluation and issue resolution

```

1 class AbstractAIBackend(ABC):
2     @property
3     @abstractmethod
4     def vector_store_manager(self) -> Optional[
5         VectorStoreManager]:
6         pass
7
8     @abstractmethod
9     def ask(self, query: str, images: List[str],
10             history: Optional[List[Dict]] = None,
11             model: ModelType = ModelType.GPT40) -> str:

```

```

11         pass
12
13     @abstractmethod
14     def activate(self) -> bool:
15         pass

```

The `AIHandler` functions as the central coordinator within the AI backend, implementing the Singleton pattern to ensure a single point of control for all AI operations. It manages:

- Registration and switching between internal AI backends within the system
- Default model selection and configuration
- Request routing and response aggregation
- Error handling and recovery

```

1 class AIHandler:
2     def __init__(self, prompt_handler=None,
3                 backends: Optional[Dict[str,
4                                     AbstractAIBackend]] = None):
5         self.__backends = backends or {
6             "openai": OpenAIHandler(prompt_handler=
7                                     prompt_handler),
8             "file_search": FileSearchHandler(
9                                     vector_store_manager=vector_store_manager)
10        }
11         self.__default_backend = list(self.__backends.keys())[0]
12         self.__default_model = ModelType.GPT40

```

Ask Implementation and Multi-Backend Support

The `AIHandler` implements a two-stage request system with the methods `ask` and `__ask_single`. While `ask` supports various request types as a public interface, `__ask_single` handles the actual backend communication.

The `ask` method supports three main scenarios:

1. single backend request
2. parallel requests to multiple backends
3. model-specific requests per backend

```
1 # Example: Multiple backends with different models
2 responses = ai_handler.ask(
3     query="Analyze text",
4     model={
5         "openai": ModelType.GPT40,
6         "file_search": ModelType.GPT4_TURBO
7     }
8 )
```

The private `_ask_single` method processes the actual request to a specific backend:

```
1 response = self._backends[backend].ask(
2     query=query,
3     images=images,
4     history=history,
5     model=model
6 )
```

This design allows handling both individual requests and multiple backend queries in parallel.

Model Integration and Flexibility

The architecture supports the integration of different AI models through the `ModelType` enum:

```
1 class ModelType(Enum):
2     GPT40 = "gpt-4o-2024-11-20"
3     GPT40_MINI = "gpt-4o-mini-2024-07-18"
4     O1 = "o1-2024-12-17"
5     O1_MINI = "o1-mini-2024-09-12"
6     O3_MINI = "o3-mini-2025-01-31"
7     DALLE3 = "dall-e-3"
8     TTS1 = "tts-1"
9     WHISPERV2 = "whisper-1"
```

OpenAI Handler

The `OpenAIHandler` provides a concrete implementation of the `AbstractAIBackend` interface, specifically designed to interact with OpenAI's suite of AI models. It manages direct communication with the OpenAI API and implements three main processing modes:

- Text generation with GPT models (GPT4O, O1, etc.)
- Image generation through DALL-E 3
- Audio processing with TTS1 and WhisperV2

The handler implements model-specific formatting and parameter management:

```

1 class OpenAIHandler(AbstractAIBackend):
2     def __init__(self, prompt_handler=None):
3         self.__api_key: Optional[str] = None
4         self.__client: Optional[OpenAI] = None
5         self.prompt_handler = prompt_handler
6         if not self.__client:
7             self.activate()
8
9     def __text_generation(self, query: str, images: List[str]
10                           ],
11                           history: Optional[List[Dict]],
12                           model: ModelType,
13                           temperature: float) -> str:
14         # Model-specific adaptations
15         if model in [ModelType.01, ModelType.01_MINI]:
16             temperature = 1 # Force temperature for 0 models
17             messages = self.__format_o_model_messages(history
18                                                         )
19         else:
20             messages = self.__format_standard_messages(
21                 history)
22
23     return self.__process_completion(messages, model,
24                                     temperature)

```

The `OpenAIHandler` standardizes error handling and response formatting across different processing modes. It maintains a single client instance for API communication to optimize resource usage and reduce latency. Additionally, it logs all requests and responses and includes automatic retry logic to handle temporary API failures.

Request Processing Pipeline

The request processing pipeline implements a structured process for processing requests, as shown in Figure 71. The pipeline starts at the client and runs through several processing stages via various handler components.

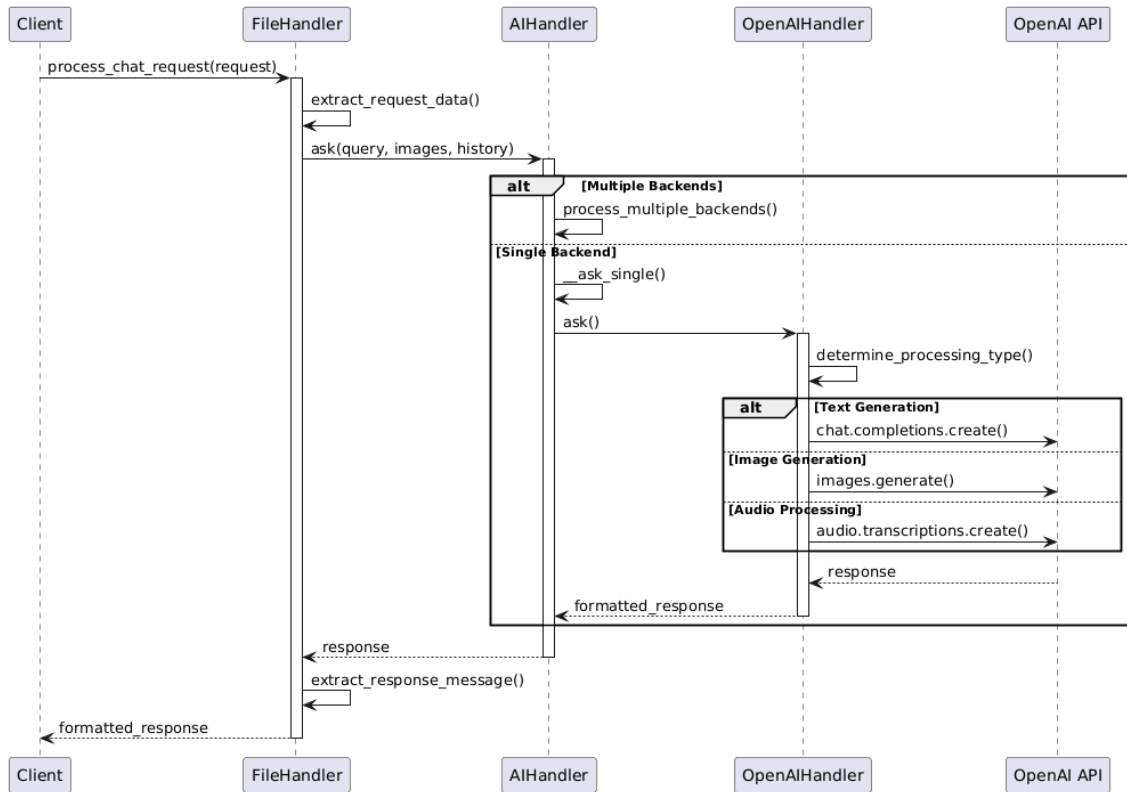


Figure 71: Request Processing Pipeline

The process starts with a request to the `FileHandler`, which serves as the first processing instance. This extracts the relevant data and forwards it to the pre-instantiated `AIHandler` object. Depending on the configuration, the system distinguishes between two main processing paths:

- **Multiple backend processing:**
 - Parallel processing by different AI backends
 - Aggregation of the results of different models
 - Coordinated response generation
- **Single backend processing:**
 - Direct forwarding to specific backend (typically `OpenAIHandler`)
 - Determination of the processing type (text, image, audio)
 - Specific API calls based on content type

The `OpenAIHandler` communicates directly with the OpenAI API and supports various processing modalities:

- text generation via chat.completions
- Image generation for visual requests
- Audio transcription for voice data

After processing, the response follows the same path back to the client, with each component carrying out its own specific post-processing. The `FileHandler` performs the final formatting and extraction of the relevant response elements.

File Handler

The `FileHandler` serves as the central entry point for processing chat requests in the AI backend system. It implements a pipeline for parsing JSON requests, handling inputs that may include text, images, and file attachments, and maintaining conversation context.

The following implementation shows the core request handling logic:

```
1 def process_chat_request(chat_request, history=None):
2     try:
3         chat_data = json.loads(chat_request)
4         message = chat_data.get('message', '')
5         images = chat_data.get('images', [])
6         files = chat_data.get('files', [])
7
8         response = forward_chat_request(
9             message=message,
10            images=images,
11            history=history
12        )
13
14        return extract_response_message(response)
15    except Exception as e:
16        logging.error(f"Error processing chat request: {e}")
17    return None
```

Context Management and Response Processing

The system manages conversation context by maintaining and utilizing previous interactions to generate contextually relevant responses. It ensures that follow-up questions receive answers that consider prior exchanges, preventing loss of context.

Additionally, response formatting is standardized across different AI models to ensure consistent output. This includes unifying response structures, aligning parameter settings, and maintaining a common schema for model outputs. The following implementation shows the core response formatting logic:

```
1 def __format_response(self, response) -> str:
2     formatted_response = {
```

```
3         "id": response.id,
4         "model": response.model,
5         "choices": [format_choice(choice) for choice in
6                     response.choices],
7         "usage": format_usage(response.usage)
8     }
    return json.dumps(formatted_response)
```

The context management system is designed to accommodate differences in formatting and parameter settings between models like O1 and GPT-4. This is achieved by mapping conversation history into model-specific input structures, standardizing prompt formats, and dynamically adjusting token limits based on model constraints.

The system ensures consistency across AI models and backend systems by tracking token usage, appending necessary metadata, and validating responses to conform to predefined output schemas. These processes allow for seamless handling of multimodal requests while preserving conversation context and maintaining uniform response quality.

7.2.2. Fine-Tuning

As previously mentioned, several LLMs by OpenAI were connected to the AI backend via API calls. Thus, interaction with said LLMs happens through inputs in a standardized format and will, likewise, result in the form of standardized outputs. The models are thereby interacted with in exactly the same state they were provided in by OpenAI, including their parameters. This comes with the drawback of only being able to interact with pre-trained models with fixed weights that may or may not be suitable equally for all tasks given.

TextVision is a tool with several capabilities and use cases that highly depend on the end user, thus rendering it difficult to pinpoint one exact use case that models could be finetuned towards. It was collectively decided that TextVision's primary selling point is the simplification of working with scientific texts. When working with scientific texts, it is a frequent goal to summarize broader sections into one cohesive summarization in order to extract the given information more efficiently. It was therefore decided to fine-tune models towards better summarization, preferably of scientific papers and articles. After thorough research and evaluation of numerous different public datasets, the dataset Scientific Lay Summarization Plos Norm from Huggingface⁴² was ultimately chosen. The dataset consists of article and summarization pairs, providing 24773 different pairs for training alone. Even though OpenAI offers various ways of fine-tuning their base models, it does come with a token-based cost. Since the dataset that was to be used for fine-tuning had already been selected beforehand, an approximation of the total token count could be deducted, revealing, under premise of utilizing the entire dataset, costs that would most likely exceed any boundaries of a given budget. It was therefore decided to instead focus on the fine-tuning of open-source models.

⁴²https://huggingface.co/datasets/pszemraj/scientific_lay_summarisation-plos-norm

The first model that was selected for fine-tuning was Mistral-7B, which was released under the Apache 2.0 license and consists of over 7 billion trainable parameters (A. Q. Jiang et al., 2023, p. 1-2). It was selected due to its moderately small parameter size, while still being able to outperform well-performing parameter-heavier open-source models like LLaMA-2-13B (Minaee et al., 2024, p. 8). Also, Mistral-7B leverages group-query attention (GQA) and sliding windows attention (SWA) which reduces the required memory and allows for a higher batch size and thus a higher throughput (A. Q. Jiang et al., 2023, p. 1).

The second selected model was Llama-3-8B, which was released under a custom license by Meta AI and consists of over 8 billion trainable parameters. Like Mistral-7B, it was selected because of its comparatively small parameter size, while it still manages to outperform many prominent, parameter-heavier models as well as generally scoring better results than Mistral-7B (Grattafiori et al., 2024, p. 3). In comparison, Llama is the newer model, being released in 2024 as opposed to 2023 for Mistral-7B (Grattafiori et al., 2024, p. 1).

The fine-tuning process itself was performed on the Pegasus HPC cluster of the DFKI⁴³. The cluster is SLURM-based, thus appropriate job scripts which specify the desired SLURM parameters were needed. Also, the cluster is container-based, using enroot⁴⁴. Thus, before any requirements can be installed and scripts can be run, an appropriate container image needs to be loaded first. These images usually come natively with their versions of Pytorch and matching NVIDIA drivers, so that the cluster GPUs can be properly utilized. The Pegasus cluster is divided into multiple partitions, each equipped with different hardware. So that the resources of the cluster could be used for the fine-tuning purposes, access to the Interactive Machine Learning (IML) partition was granted. The slurm batch script that was used to train the Mistral model can be seen down below.

```

1 #!/bin/bash
2 #SBATCH --job-name=finetuning_mistral
3 #SBATCH --output=job_output.txt
4 #SBATCH --error=job_error.txt
5 #SBATCH --ntasks=1
6 #SBATCH --cpus-per-task=4
7 #SBATCH --gres=gpu:1
8 #SBATCH --time=12:00:00
9 #SBATCH --partition=A100-IML
10 #SBATCH --mem=128G
11
12 NUM_GPUS=1
13
14 echo "Starting job on node: $(hostname)"

```

⁴³<https://pegasus.dfki.de/>

⁴⁴<https://github.com/NVIDIA/enroot>

```

15 echo "Number of GPUs: ${NUM_GPUS}"
16 echo "Current directory: $(pwd)"
17
18 unset MASTER_ADDR
19 unset MASTER_PORT
20
21 export MASTER_ADDR=$(hostname)
22 export MASTER_PORT=29500
23
24 echo "MASTER_ADDR: $MASTER_ADDR"
25 echo "MASTER_PORT: $MASTER_PORT"
26 echo "SLURM_PROCID: $SLURM_PROCID"
27 echo "SLURM_NTASKS: $SLURM_NTASKS"
28
29 srun \
30   --container-image=/enroot/nvcr.io_nvidia_pytorch_23.12-py3.
   sqsh \
31   --container-workdir="$(pwd)" \
32   --task-prolog="$(pwd)/install.sh" \
33   --container-mounts="/netscratch/dehlers:/netscratch/dehlers
   " \
34   torchrun \
35     --nproc_per_node=${NUM_GPUS} \
36     --master_addr=$MASTER_ADDR \
37     --master_port=$MASTER_PORT \
38     finetuning_mistral_lora.py

```

It can be noticed that this script called `install.sh` runs a second script before it starts the actual fine-tuning script. The `install.sh` script handles necessary dependencies using `pip` by, among others, installing libraries such as `pytorch`, `huggingface-hub`, `datasets`, `transformers` and `peft`.

The training script itself was written in Python, using Pytorch as well as importing the `transformers` library. During runtime, a connection to the Huggingface hub is created, using an access token. The tokenizer and model are then directly loaded. Since LoRA was used for the fine-tuning process, also the config for the LoRA adapters was set up. After setup, the model was then adjusted with a multitude of training parameters and ultimately trained. Down below is a snippet of the fine-tuning script that shows the function that loads the model and the appropriate tokenizer as well as setting up the LoRA config. The entire training script can be found under A.4.

```

1 def load_model_and_tokenizer(local_rank, world_size):
2     tokenizer = AutoTokenizer.from_pretrained("mistralai/
   Mistral-7B-Instruct-v0.2", use_fast=True)
3     if tokenizer.pad_token is None:

```

```

4         tokenizer.pad_token = tokenizer.eos_token
5
6     base_model = AutoModelForCausalLM.from_pretrained(
7         "mistralai/Mistral-7B-Instruct-v0.2",
8         torch_dtype=torch.bfloat16,
9         low_cpu_mem_usage=True
10    )
11
12    if hasattr(base_model, "gradient_checkpointing_enable"):
13        print("Enabling gradient checkpointing on base model
14              before DDP wrapping...")
15        base_model.gradient_checkpointing_enable()
16
17    base_model = prepare_model_for_kbit_training(base_model)
18
19    lora_config = LoraConfig(
20        task_type=TaskType.CAUSAL_LM,
21        r=16,
22        lora_alpha=32,
23        target_modules=["q_proj", "v_proj"],
24        lora_dropout=0.05,
25        bias="none"
26    )
27    model = get_peft_model(base_model, lora_config)
28    model.train()
29    model = model.to(f"cuda:{local_rank}")
30
31    if world_size > 1:
32        model = DDP(model, device_ids=[local_rank],
33                    output_device=local_rank, find_unused_parameters=
34                    False)
35
36    return model, tokenizer

```

A problem that arose during the beginning of the fine-tuning process was the context window size of the selected models. Regarding language models, a context window refers to the number of input tokens that the model is able process at once when generating a response. Exceeding that window can lead to unexpected behavior and deterioration in performance (Jin et al., 2024, p. 1), as previous tokens that fell out of the window's range are forgotten by the model. Both selected models have a context window of 8192 tokens, which would be a sufficient size to process most basic instruction tasks but would be too limiting in the context of processing long scientific articles. Analyzing the training data revealed an average of 10133 tokens per article in the dataset, with the longest article even being 37278 tokens long. Given that the average of articles already has a token length

that exceeds the context window by almost 2000 tokens, it was decided that a context size of 8192 would not be sufficient to effectively utilize the training data. In response, two slightly altered versions of the original models were selected to be trained instead. This includes the following two models:

- **Mistral-7B-Instruct-v2**: An already finetuned version of Mistral-7B with a context window size of 16384. The model was finetuned towards instruction purposes, which complements its role in TextVision as an AI assistant ⁴⁵.
- **Llama-3-8B-16k**: A user-modified version of the original Llama-3-8B model which comes with a context window size of 16384⁴⁶.

As was mentioned, some articles would even exceed the new context window size. Those articles were automatically excluded from the training process during runtime, as truncation would result in information loss that is detrimental in a summarization context. In order to get the models to actually differentiate between articles and summaries, special tokens were inserted into the input, separating summaries from their respective article. It was then made sure that model only predicts the summaries during the training process, by setting token values before the inserted summary token to -100. For the training process itself, a multitude of adjustable hyperparameters impact the final result. The most relevant hyperparameters are:

- **Epochs**: Number of training iterations, with one epoch referring to the model processing the entirety of the training data. The best results could be achieved for an epoch number of 4. The results stagnated for higher epochs count, which would thus only unnecessarily prolong the training time.
- **Learning rate**: Determines how fast the model weights are updated during the training process. Here a comparatively small learning rate of 2e-5 was chosen and yielded the best results, as high learning rates could potentially lead to the model overwriting pre-trained knowledge too quickly.
- **Batch Size**: Refers to the number of samples that can be processed at once. A sample in this case refers to an article/summary pair. To not exceed the available VRAM on the cluster, a batch size of 4 was chosen. This was complemented by another hyperparameter, the gradient accumulation steps.
- **Gradient Accumulation Steps**: With gradient accumulation, the model does not update the weights after each batch. It computes the gradients for several mini-batches and accumulates them instead. Weight updates are only conducted after multiple mini-batches have been processed, which results in a larger effective batch size with lower required memory. The best results could be achieved for a gradient accumulation step size of 16. In combination with a batch size of 4, which means that each step processes 4 samples and gradient updates happen every 16 steps, 4 x

⁴⁵<https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2>

⁴⁶<https://huggingface.co/mattshumer/Llama-3-8B-16K>

16 = 64 samples contribute to a single weight update. The GPU thus only processes 4 samples each step, but the model behaves as if it would with a batch size of 64.

- **Optimizer:** The optimizer is the algorithm that determines how loss is calculated and how the model weights are updated. During the finetuning process, the optimizer `adamw_torch` was selected, which is the pytorch version of AdamW ⁴⁷, an optimizer that combines an updatable learning rate with decoupled weight decay to prevent the weight decay from interfering with the actual weight updates.

After each finished training iteration, the performances of the models were evaluated, using a separate script, which can be found under A.5. The fine-tuned models were evaluated on the test data of the dataset and compared to the results of its respective base model. As evaluation metrics the scores ROUGE-1, ROUGE-2, ROUGE-L and ROUGE-LSUM were chosen, as they are measured by similarities in generated and ground-truth summary. The ROUGE-1 score is defined as

$$\text{ROUGE-1} = \frac{\text{Number of overlapping unigrams}}{\text{Total unigrams in reference text}}. \quad (1)$$

The ROUGE-2 score is defined as

$$\text{ROUGE-2} = \frac{\text{Number of overlapping bigrams}}{\text{Total bigrams in reference text}}. \quad (2)$$

The score ROUGE-L is defined as

$$\text{ROUGE-L} = \frac{\text{LCS length}}{\text{Total words in reference text}}. \quad (3)$$

The ROUGE-LSUM score takes the longest common subsequences but computes it over the entire summary. It is defined as

$$\text{ROUGE-LSum} = \frac{\text{LCS length across summary}}{\text{Total words in reference summary}} \quad (\text{C.-Y. Lin, 2004, p. 1-2}). \quad (4)$$

After training was completed, the best achieved results were benchmarked against their base model counterparts to determine how big of an impact the fine-tuning process had. The best results were:

Table 13: Results Fine Tuning

Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-LSUM
Mistral-7b-instruct Base	0.42	0.14	0.22	0.22
Mistral-7b-instruct Fine-tuned	0.46	0.15	0.24	0.24
Llama-3-8b-16k Base	0.35	0.10	0.17	0.18
Llama-3-8b-16k Fine-tuned	0.37	0.11	0.18	0.18

⁴⁷<https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html>

It can be seen that there is generally a slight increase in ROUGE scores, thus the fine-tuning process can ultimately be considered successful, though it should be noted that the results were not benchmarked against other models, primarily the OpenAI models, to properly determine whether it would be advantageous to use the fine-tuned open-source models rather than the OpenAI models when summarization of scientific papers is the goal.

7.2.3. Prompt management

The Prompt Management System handles, evaluates, and optimizes AI prompts. The system’s `PromptHandler` class provides prompt management features including evaluation processes based on research papers, prompt optimization, and suggestion generation.

Prompt Handler

Figure 72 illustrates the core components and their relationships in the Prompt Management System.

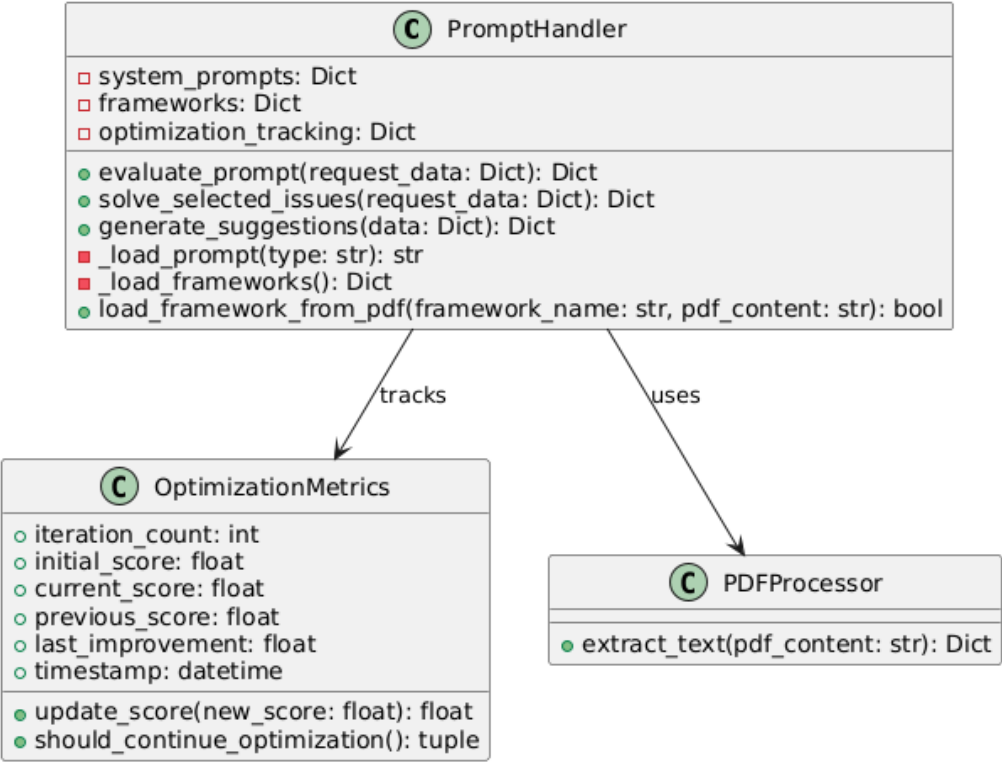


Figure 72: Prompt Management System Architecture

The `PromptHandler` serves as the central component, managing:

- System prompts repository

- Framework management
- Optimization tracking
- Suggestion generation

Dynamic Framework Integration

The system processes PDF documents containing prompt engineering papers, which serve as evaluation frameworks. These documents are initially uploaded to the Node.js backend, where their contents are extracted and stored in MongoDB. When needed, the Node.js backend retrieves and forwards the document contents to the Python AI backend, which performs the actual processing and analysis.

Since the logic for handling and evaluating frameworks resides in the AI backend, this section focuses on its processing mechanisms. This structure enables updates to evaluation criteria and framework switching without modifying the overall system architecture.

The following implementation shows how the AI backend extracts text from an uploaded PDF document and makes it available for further processing:

```
1 def load_framework_from_pdf(self, framework_name: str,
2                               pdf_content: str) -> bool:
3     pdf_processor = PDFProcessor()
4     extraction_result = pdf_processor.extract_text(
5         pdf_content)
6     return extraction_result["success"]
```

In this implementation, the `PDFProcessor` extracts text from a given PDF document. The extracted content is then used to generate evaluation criteria dynamically, allowing for adaptable framework integration.

Prompt Evaluation and Optimization Pipeline

The system implements a two-stage process for prompt evaluation and optimization, as illustrated in Figure 73. This process uses PDF-based frameworks as evaluation criteria and enables systematic prompt improvement through iterative assessment and optimization steps.

The evaluation and optimization process consists of two main stages:

Stage 1: Evaluation Process

The evaluation stage begins with loading the framework from a provided PDF using `load_framework_from_pdf()`, which extracts and processes the framework guidelines. The `evaluate_prompt()` method then performs a comprehensive analysis: As shown in the workflow diagram 73, this stage:

1. Loads and processes framework guidelines from PDF
2. Analyzes the prompt against these guidelines
3. Identifies specific issues and their severity levels
4. Calculates a comprehensive quality score
5. Generates detailed evaluation results

Stage 2: Optimization System

The optimization stage focuses on resolving selected issues identified during evaluation. The system employs the `OptimizationMetrics` dataclass to track improvement progress. The `solve_selected_issues()` method processes only those issues specifically selected by the user through the frontend. This targeted approach allows for precise optimization based on user priorities. The optimization workflow:

1. Receives selected issues from the frontend via Node.js backend
2. Generates specific solutions using the `solve_issues` system prompt template
3. Implements improvements while preserving the prompt's original intent
4. Re-evaluates the optimized prompt

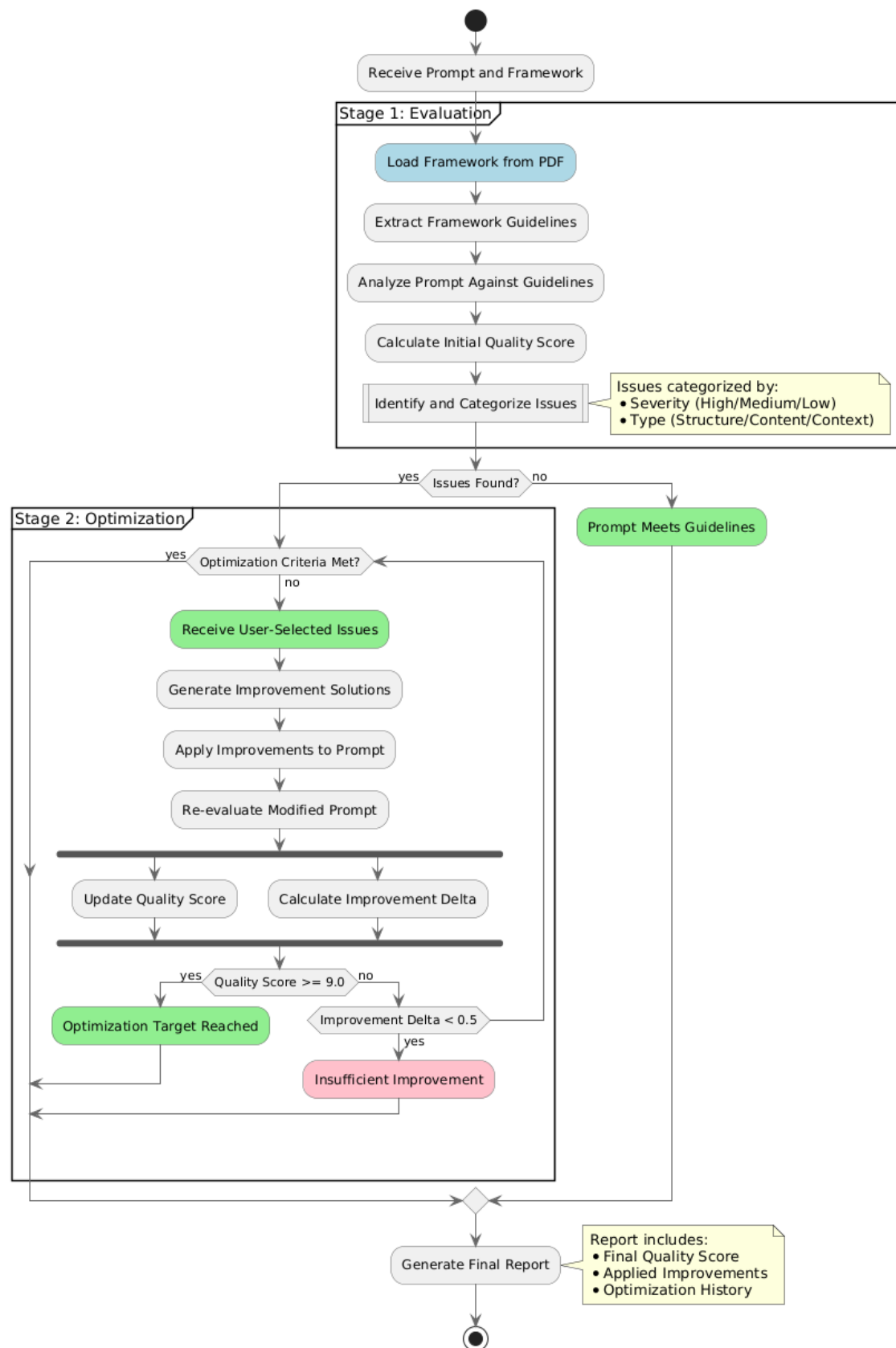


Figure 73: Prompt Evaluation and Optimization Workflow

The process continues until meeting one of these termination conditions:

- Achievement of target quality (score ≥ 9.0)
- Insufficient improvement ($\Delta < 0.5$)
- Resolution of all critical issues

This two-stage approach ensures thorough evaluation against framework guidelines while providing flexible, user-directed optimization focusing on selected issues. The separation of evaluation and optimization stages allows for precise control over the improvement process while maintaining efficiency and effectiveness.

Suggestion Generation

The suggestion generation system provides context-aware recommendations by analyzing user input, past interactions, and any relevant external documents. Instead of providing direct answers, the system generates refined prompts that encourage deeper exploration of the topic while ensuring continuity in the conversation. This approach helps guide the user through a structured inquiry process and prevents redundant or irrelevant suggestions. The system operates in two distinct modes, as illustrated in Figure 74.

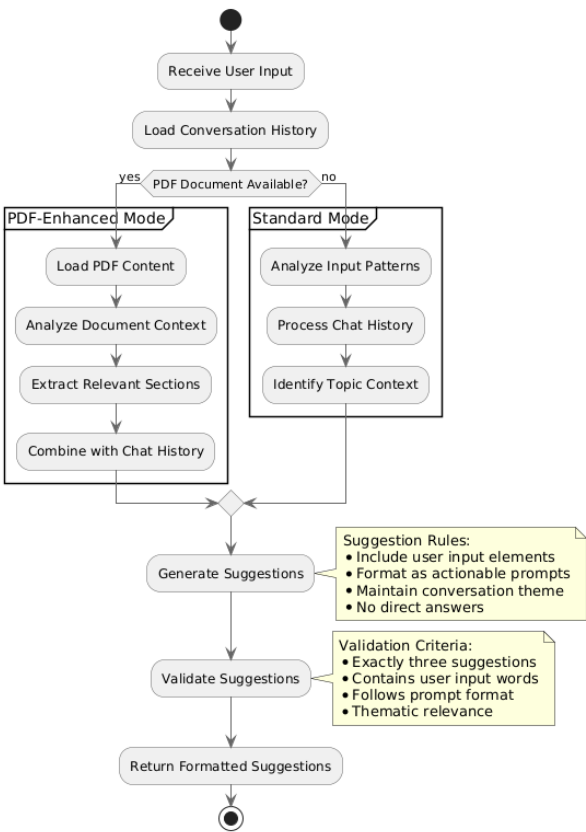


Figure 74: Suggestion Generation Workflow

PDF-Enhanced Suggestions

When a PDF document is provided, the system uses the `generate_suggestions_with_pdf` template. This mode includes document content as context and analyzes:

- The relationship between the current user input and the PDF content
- Relevant sections from the PDF that align with the user's focus
- Chat history patterns in conjunction with document themes

Standard Suggestions

In the absence of PDF content, the system employs the `generate_suggestions_without_pdf` template, which focuses on:

- Pattern recognition in current user input
- Analysis of conversation flow and context
- Historical interaction patterns to predict user intent

This approach ensures that the generated suggestions are based on the user's prior engagement with the system and not solely on the most recent query, thereby maintaining contextual relevance.

Both modes strictly adhere to key principles:

1. Always generate exactly three suggestions
2. Format suggestions as actionable prompts or instructions
3. Never provide direct answers to the user's current query
4. Maintain thematic relevance to the current conversation
5. Each suggestion must contain elements from the user's input

For example, if a user asks "What are the key points of machine learning?", the system might generate:

- "Explain how machine learning algorithms process data"
- "Compare different types of machine learning approaches"
- "Analyze the evolution of machine learning techniques"

Rather than providing direct answers, these suggestions reformulate the user's interest into new, focused prompts that maintain the conversation's direction and potentially explore different aspects of the topic.

7.2.4. File Processing

The file processing system implements a pipeline for handling multimodal data, particularly focusing on document processing, vector storage, and multimedia content analysis. The system architecture comprises four main components: **PDFProcessor**, **FileSearchHandler**, **VectorStoreManager**, and **FileHandler**, working in concert to provide comprehensive file processing capabilities (see Figure 75).

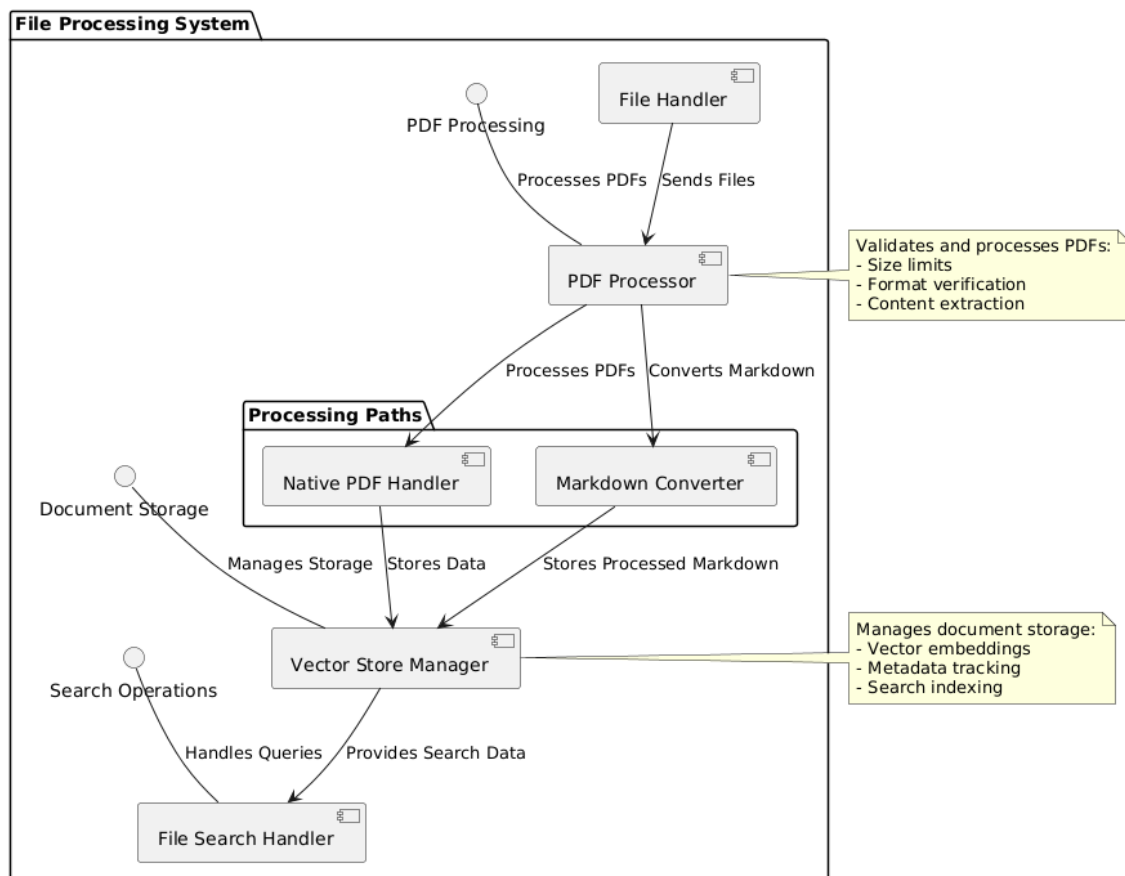


Figure 75: File Processing System

PDF Processor Component

The PDF Processor serves as a crucial bridge between incoming PDF documents and Language Learning Models (LLMs) in the AI backend system. It manages three essential operations:

1. PDF validation and integrity checks
2. Text extraction and processing

3. Binary conversion for different LLM requirements

The processor supports two primary processing pathways:

- **Native PDF Processing:** Direct binary conversion for modern LLMs that can process raw document structures.
- **Markdown Conversion:** Extracting and formatting text into structured Markdown for legacy systems that require preprocessed text input.

All processing results are returned in a standardized JSON format to ensure consistent downstream handling. The following structure represents a typical output:

```
1 {  
2     "success": bool,  
3     "content": {  
4         "pages": [{"page_number": int, "text": str, "metadata  
5             ": dict}],  
6         "total_pages": int  
7     }  
}
```

This structured output allows downstream AI components to efficiently process extracted text, maintain document structure, and incorporate metadata for improved contextual understanding.

The PDF Processor integrates with other system components, including the File Handler, Vector Store Manager, Prompt Handler, and File Search Handler. Through these integrations and its direct pathways to AI model APIs, the processor enables efficient document processing workflows and content delivery to target LLMs. This comprehensive integration approach ensures maximum flexibility in document handling, whether providing markdown-formatted text for models without native PDF support or delivering binary PDF data to compatible AI endpoints.

File Search and Vector Stores

The AI backend implements a document retrieval and search system that combines File Search Handler and Vector Store Manager components. This system enables AI models to access external knowledge sources, including documentation and user-provided documents, complementing their base knowledge.

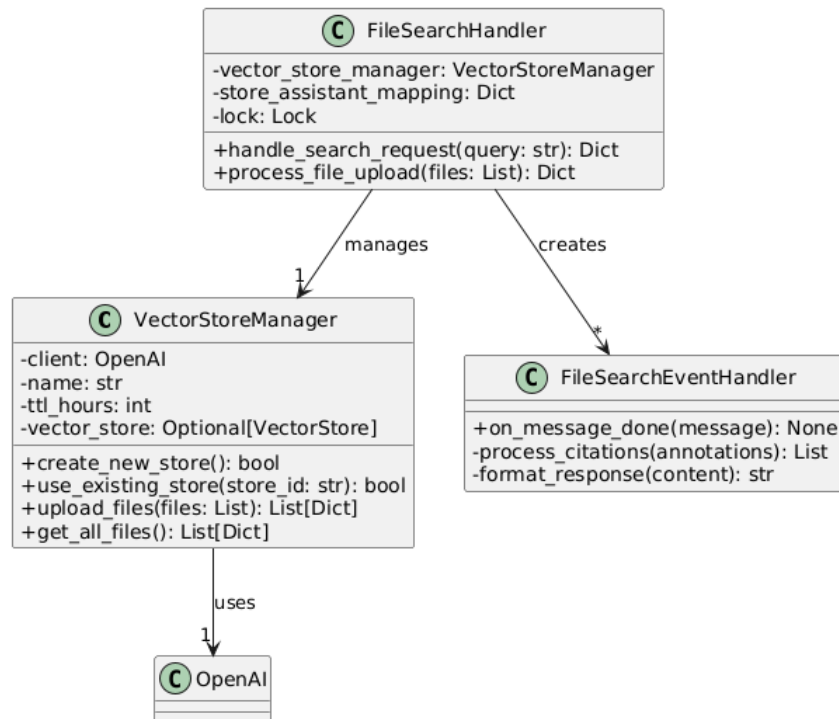


Figure 76: Vector Store Architecture showing component relationships

As illustrated in Figure 76, the system architecture comprises three main components:

1. File Search Handler

- Manages search operations via OpenAI's Assistant API
- Ensures thread-safe operations through lock management
- Supports both new and existing assistant scenarios
- Integrates with Vector Store Manager for document access

2. Vector Store Manager

- Controls document storage and retrieval operations
- Implements TTL-based resource management
- Provides comprehensive status tracking
- Enables flexible store creation and access

3. File Search Event Handler

- Processes real-time citations
- Manages document references
- Ensures academic rigor in responses

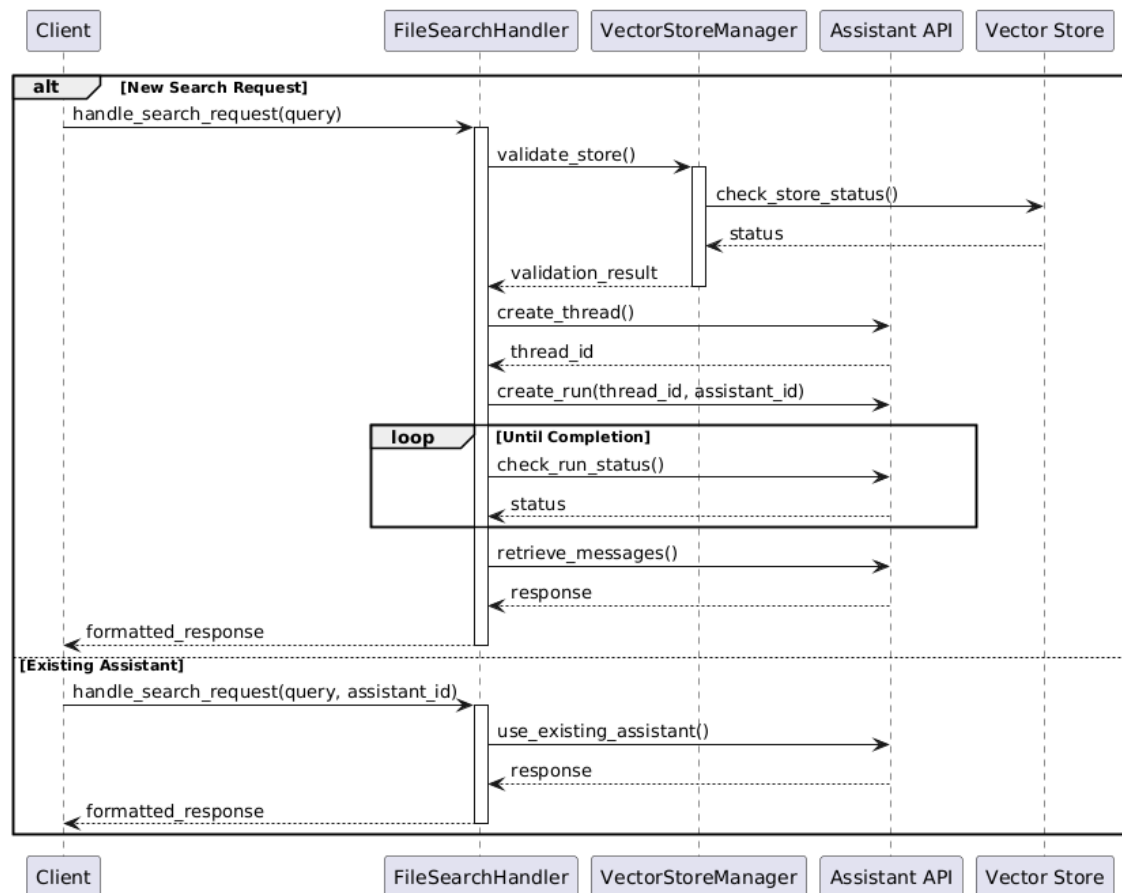


Figure 77: Search Operations Workflow

Figure 77 demonstrates the search request workflow, showing how the system handles both new search requests and operations with existing assistants. The sequence diagram illustrates the interaction between components during search operations, including:

- Initial request validation
- Thread and assistant management
- Status monitoring
- Response formatting and delivery

The handler processes search requests through a dedicated pipeline that supports both new and existing assistants. It maintains thread safety through careful lock management and provides comprehensive error handling. Search operations can be initiated with either a `vector store` ID or an existing `assistant` ID, offering flexibility in how documents are accessed and searched. This flexibility is enhanced by the ability to attach vector stores to

both Assistants and Threads, enabling contextual document access. The manager implements methods for both creating new stores and accessing existing ones, ensuring efficient resource utilization and data consistency across the system. Through the integration of the File Search Handler and Vector Store Manager, the system provides robust document management and retrieval capabilities. This design enables centralized document control while maintaining search flexibility, with automatic resource management and status monitoring. This integration enables document-based AI interactions, ranging from basic searches to assistant-based operations.

The system supports two primary search paths:

1. New Search Requests

- Vector store validation
- Thread creation
- Assistant configuration
- Response monitoring and retrieval

2. Existing Assistant Operations

- Direct assistant utilization
- Streamlined processing
- Immediate response generation

Citation Handling

Citation handling is integrated throughout the search process, ensuring proper attribution and maintaining academic standards. This is used to indicate which information comes from which document. This is particularly important when working with large numbers of documents. The handler processes citations in real time as messages are completed, ensuring that all references to source documents are properly tracked and presented. This maintains academic rigour by clearly linking statements to their source documents and providing a clear citation trail for all information.

7.3. Tests

The testing strategy for the Python AI backend focuses on validating the correctness and robustness of core components, rather than achieving comprehensive code coverage. Emphasis is placed on critical functionalities that directly affect the reliability and user-facing behaviour of the system, such as model interaction, document processing, authentication, and concurrency control. Integration with external APIs, in particular OpenAI, is simulated via mocking to ensure consistent and reliable test execution.

7.3.1. Test Environment Structure

The test suite is organized within the `tests/` directory and includes unit and integration tests. The structure is as follows:

```
tests/
  test_files/
    sample_lecture_slides/
      01_Orga_WS_23_24_IT_Controlling.pdf
      ...
      11_Klausurvorbereitung_WS_23_24_IT_Controlling.pdf
    prompt-pattern.pdf
    sample.pdf
    sample_pdfprocessor.pdf
  conftest.py
  test_ai_handler.py
  test_auth.py
  test_chat_history.py
  test_concurrent_file_search_handler_mock.py
  test_file_search.py
  test_file_search_assistant.py
  test_main_endpoints.py
  test_openai_handler.py
  test_pdf_processor.py
  test_prompt_handler.py
  test_prompt_handler_integration.py
  test_vector_store_management.py
```

7.3.2. Test Overview

- **test_ai_handler.py**
Verifies the AIHandler component, covering support for multiple OpenAI models (e.g., GPT-4o, O1), backend switching, prompt evaluation and optimization, and error handling. Mocking is used to simulate OpenAI responses.

- **test_auth.py**
Validates the authentication system based on HTTP Basic Auth, ensuring proper handling of valid and invalid credentials and correct configuration loading.
- **test_chat_history.py**
Tests how conversation history is maintained and passed to different models. It ensures proper ordering, model-specific formatting, and resilience to invalid or empty input.
- **test_concurrent_file_search_handler_mock.py**
Focuses on concurrency and thread safety within the FileSearchHandler, verifying that shared resources are correctly managed during concurrent access and that the initialization of the wizard is stable under load.
- **test_file_search.py**
Simulates real-world end-user interactions with the document retrieval system through the high-level AIHandler interface. Ensures that document loading and question answering behaves as expected.
- **test_file_search_assistant.py**
Targets the internal implementation of document search. Tests include vector store and assistant management, concurrent document processing, and OpenAI Assistants API usage.
- **test_main_endpoints.py**
Verifies the behavior of the FastAPI endpoints, including the correct generation of HTTP responses, integration with back-end components, and error handling.
- **test_openai_handler.py**
Verifies formatting and processing logic for sending requests to OpenAI APIs, including model-specific parameters and image/audio support. Also checks for proper error handling when APIs fail.
- **test_pdf_processor.py**
Ensures the correct operation of the PDFProcessor component, including PDF validation, extraction, performance constraints, and handling of edge cases such as oversized or malformed files.
- **test_prompt_handler.py**
Tests the PromptHandler's functionality in managing and optimizing prompts. Focus lies on prompt scoring, issue detection, and the ability to improve prompts via simulated AI responses.
- **test_prompt_handler_integration.py**
Provides end-to-end tests for the prompt engineering system using live OpenAI API calls. It validates the full optimization pipeline from framework-based evaluation to improvement iteration.

- **test_vector_store_management.py**

Ensures that vector store operations (create, upload, list, remove) are correctly implemented. These tests validate document persistence and assistant access to vectorized content.

7.3.3. Test Execution

All tests can be executed using `pytest`, either individually, by file, or as an entire test suite. It is important to ensure that the working directory and import paths are correctly set before running the tests. This can be achieved either by providing explicit relative paths to test files or by manually navigating into the correct subdirectory (e.g., `ki_api`) before execution.

Examples for running tests:

```
# Run all tests in the entire project (from project root)
pytest

# Run all tests in a specific folder (from project root)
pytest ki_api/tests/

# Run a specific test file
pytest ki_api/tests/test_ai_handler.py

# Run a single test function with verbose output
pytest -v ki_api/tests/test_ai_handler.py::TestAIHandler::test_gpt4o_model
```

Alternatively, you can navigate directly into the test directory and execute tests relative to that context:

```
PS C:\Users\marlo\PycharmProjects\pgaim-ai-backend> cd ki_api
(.venv) PS C:\Users\marlo\PycharmProjects\pgaim-ai-backend\ki_api> pytest tests/
```

To measure test coverage during execution, the `pytest-cov` plugin can be used. Example:

```
pytest --cov=ki_api ki_api/tests/
```

This will provide detailed coverage statistics for all modules inside the `ki_api` package. For consistent results, make sure the virtual environment is enabled and all required dependencies are installed.

The test infrastructure is configured using `pytest.ini` and supports asynchronous test execution, integration markers, and modular test organization. External dependencies and APIs are consistently mocked using `unittest.mock` to ensure reliable and repeatable results.

In summary, the testing approach focuses on verifying the correctness of essential system components rather than achieving full coverage. The combination of unit and integration

testing ensures that key functionalities-such as model interaction, file processing, prompt evaluation, and concurrency handling-are validated under realistic and critical conditions. Further extensions to the testing strategy would be possible, including more detailed code coverage analysis, performance benchmarks, and automated execution within CI/CD pipelines. These enhancements could provide additional insights during long-term operations and ongoing development.

7.4. Known Problems

The Python AI backend, while robust in many aspects, exhibits several limitations and issues that users should be aware of. This section provides a comprehensive overview of these challenges to help users navigate potential pitfalls and understand ongoing development efforts.

7.4.1. Performance Considerations

Response speed across the AI pipeline remains an area for improvement. The RealTime API potentially offers faster processing compared to the standard implementation. Users working with time-sensitive applications should consider this performance differential when designing their workflows.

Large-scale operations, particularly those involving extensive document collections, may experience performance degradation. The vector store exhibits slower search capabilities when handling substantial document volumes, coupled with high memory consumption during vector calculations.

Under high concurrent loads, the system may experience performance issues due to potential race conditions in vector store management and deadlocks in multi-backend scenarios. We have partially mitigated these concerns through request queuing implementation, though users should remain mindful of these limitations in high-traffic environments.

7.4.2. Model Behavior and Limitations

The inherent nature of LLMs introduces certain inconsistencies, particularly evident during prompt evaluation. Users may observe slight variations when rating identical prompts multiple times. This is a characteristic behavior of current LLM technology rather than an implementation flaw.

Temperature parameter handling presents another challenge, as it is not consistently applied across all models. Notably, O1 models force temperature = 1, effectively overriding user settings. We have implemented automatic temperature adjustment for these models as a workaround, though users should recognize this as intended behavior rather than a defect.

Hallucination Management

The system implements targeted mechanisms to mitigate hallucinations within generated content. At the technical level, the document-based knowledge retrieval system leverages vector store implementation (RAG) to ground responses in verifiable content, while automatic citation and source attribution ensures factual accountability. Temperature control (default 0.7) with user adjustment capabilities balances creativity and factuality, supplemented by JSON-based structured response formats that enforce consistency.

These technical approaches are complemented by prompt engineering strategies, including explicit citation requirements in system instructions, academic rigor enforcement in assistant definitions, and framework-based evaluation criteria. The system mandates source referencing in knowledge-based responses and implements verification procedures through:

- Clear role definition for scholarly and factual outputs
- Specific formatting requirements for structured outputs
- Explicit prohibitions against providing unsourced answers
- Severity classification for different types of factual issues
- Character limits enforcing concise, factual responses
- Input word inclusion requirements maintaining contextual relevance

Despite these measures, users should remain aware that hallucination management represents an ongoing challenge rather than a completely solved problem.

The system also faces challenges with context management, including:

- Limited size of context windows (without File Search)
- Maximum chunk constraints with file search
- Potential for hallucinations in generated content
- Suggestion systems that may not always capture user intention, particularly during cold starts

7.4.3. Token and Memory Management

Token limit management requires attention, as the system lacks automatic shortening of the context window. This limitation creates a risk of exceeding model token limits, especially with large conversation histories. Users must manually manage their context windows until planned enhancements address this issue.

Memory management presents several challenges in production environments:

- Memory leaks may occur in long-running sessions
- Temporary files may not be efficiently cleaned up
- Large vector stores can lead to resource consumption issues

7.4.4. Document Processing Limitations

PDF processing exhibits several notable limitations:

- Memory issues when handling large PDF files (exceeding 50MB)
- Incomplete text extraction from complex layouts featuring numerous images, illustrations, nested tables, or handwritten content
- OCR limitations when processing scanned documents

As a workaround, we recommend adhering to file size limits and implementing pre-processing steps for complex documents. Alternatively, the size limit can be raised.

File type support remains limited for certain formats (PDF, images, audio).

7.4.5. Technical Implementation Issues

JSON processing introduces unnecessary escaping of valid symbols, particularly single quotes. For example, valid JSON containing "text with ' quote" may be incorrectly escaped as "text with ´quote". A manual JSON cleaning mechanism is in place to mitigate this issue.

API compatibility requires ongoing attention due to:

- Version conflicts with OpenAI API updates
- Breaking changes in model response formats
- Backward compatibility challenges

Our current approach involves version pinning and compatibility layers, with continuous monitoring and updates to address emerging issues.

Response formatting inconsistencies manifest across different models, including HTML tag handling variations and special character encoding issues. While we have implemented standardized response parsing, further standardization efforts are needed.

Model switching operations occasionally exhibit lagging, context loss during backend changes, and incomplete cleanup of previous sessions. A forced context reset on model switch serves as a temporary workaround, with a more comprehensive architecture redesign planned.

Request timeout handling shows inconsistent behavior across components, with long-running requests not properly terminated and missing timeout configurations for certain operations.

These known issues should be carefully considered in any future development efforts. Addressing them will be essential to enhance the reliability, performance, and functionality of the Python AI backend. Future development teams should prioritize these challenges to ensure the system reaches its full capability.

8. Design & Procedure of the User Study

This chapter outlines the methodology and structure of the user study conducted to evaluate the efficiency and usability of the TextVision interface in comparison to a traditional word processing tool like Microsoft Word. The study was designed to assess how TextVision's integrated features impact document creation, particularly focusing on prompt-based interactions and cognitive workload. By employing both qualitative and quantitative methods, the study aims to provide insights into user experience, usability, and efficiency.

8.1. Research Objectives and Hypotheses

The primary goal of this study is to determine whether the TextVision interface improves document creation efficiency, usability, and overall user experience compared to Microsoft Word. This investigation is guided by the following research question:

Research Question: *"Does the TextVision interface improve the efficiency, usability, and overall user experience of document creation compared to traditional word processing tools?"*

Given that prompt-based interactions are a core component of TextVision, an additional sub-question was formulated to examine their specific impact on cognitive load:

Sub-Question: *"In what ways do the PromptDesigner and Prompt Recommendation features reduce cognitive load during the document creation process?"*

This sub-question was introduced because prompting plays a central role in our developed tool, allowing users to e.g. receive Prompt Recommendations or optimize/save their own prompts in the PromptDesigner window. Since traditional word processors such as Microsoft Word tend to lack such an integrated prompting system, it is crucial to investigate whether these features contribute to a more seamless and less cognitively demanding workflow. Based on this, the following hypotheses were formulated:

1. TextVision helps users create documents faster and more easily than Microsoft Word.
2. The integration of Prompt Recommendations and the PromptDesigner reduces cognitive load compared to traditional word processing tools that do not incorporate such prompt features.
3. The interactive features, including the AI-Assistant and voice commands, enhance the usability compared to traditional tools.

By testing these hypotheses, the study aims to provide empirical evidence on the benefits and challenges of using an AI-enhanced document editing tool. The subsequent sections will detail the study's design, participant distribution, and execution methodology.

8.2. Planning of the User Study

Before conducting the user study, it was necessary to obtain ethical approval to ensure compliance with research guidelines. Initially, approval at the University of Oldenburg was considered, but feedback from the previous project group indicated that the approval process there is significantly longer. In order to avoid unnecessary delays, it was necessary to opt for the fast-track approval procedure offered by the DFKI Ethics Committee, which allowed for a more efficient procedure. The application for ethical approval, which can be found in the Appendix A.1.1, includes details on the study's general information, motivation, research procedure, and ethical considerations.

8.2.1. Recruitment of Participants and Grouping

Unlike studies targeting specific professional domains, the user study was designed to reflect the broad applicability of TextVision in everyday document-related tasks, where advanced technical knowledge is not a prerequisite. As a result, it did not impose many requirements beyond the fundamental one that participants had to be at least 18 years old.

A number of strategies were used to recruit participants. The user study was advertised on the bulletin board of Stud.IP and in various Discord channels, using a promotional text (see Appendix A.1.2). Additionally, past university lecturers were contacted via email, requesting them to promote the study in their lectures, and acquaintances were also contacted.

Experts generally recommend having five to ten participants for a user study (Caine, 2016). Figure 78 showcases a scatter plot of the distribution of different sample sizes used in user studies.

Based on this, the initial goal was to recruit a total of 20 participants, aiming for a balanced distribution of 10 people per skill group. Participants were classified into two groups: low-level and high-level, based on their previous experience and familiarity with AI tools. During the recruitment process, 27 individuals expressed interest, but some had to withdraw, leaving 19 participants.

Interestingly, during the study, one to two participants originally classified as low-skill users adapted quickly to TextVision, even outperforming some high-skill participants. Despite having little prior experience with AI tools, they demonstrated a natural affinity for the system and learned to use it efficiently within a short time. This suggests that intuitive AI-based interfaces like TextVision can significantly lower the entry barrier for users unfamiliar with AI tools.

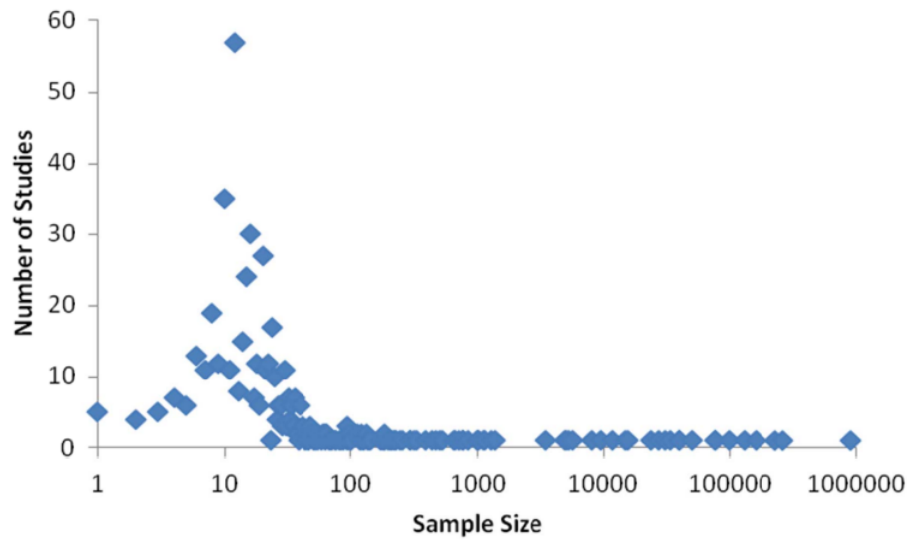


Figure 78: Scatter plot of varying sample sizes in user studies (Caine, 2016)

8.2.2. Questionnaire Selection

To comprehensively assess usability, efficiency, and cognitive workload, standardized quantitative questionnaires were combined with a qualitative semi-structured interview. Each measurement instrument was chosen based on its relevance to the study's objectives.

The *Basic Questionnaire* (see Appendix A.1.3) was filled out before participants engaged with TextVision. It was designed to gather insights into their prior experience with LLMs and traditional word processors. This helped gauge participants' prior experience with AI tools and word processors, allowing for a comparison of their initial familiarity with their experiences after using TextVision.

After the participants used TextVision and worked on their study tasks, they had to complete two different questionnaires; the *System Usability Scale* (SUS)⁴⁸ (see Appendix A.1.4) and *NASA-TLX*⁴⁹ (see Appendix A.1.5). SUS is a widely used, standardized tool for assessing perceived usability and was chosen because it provides a quick yet reliable measure of how users experience an interface. Given that TextVision integrates multiple different components (document editor, AI-Assistant, PDF viewer, voice commands) and features, it was essential to quantify how seamless and user-friendly participants found the system as a whole. To evaluate the cognitive workload, the NASA TLX questionnaire was used. Since TextVision introduces AI-assisted workflows, it was important to determine whether its features reduced or increased mental effort compared to traditional word processing tools. NASA-TLX, which measures workload across six dimensions (Mental

⁴⁸SUS PDF Generator. (n. d.). <https://pdf.sus.tools/>

⁴⁹Hart and Staveland. (n. d.). NASA Task Load Index. <https://humansystems.arc.nasa.gov/groups/tlx/downloads/TLXScale.pdf>

Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration), was suitable for identifying whether AI-driven assistance reduced cognitive strain or introduced new challenges.

In addition to these questionnaires, a *semi-structured interview* was also conducted at the end to gather qualitative feedback. The interview questions (see Appendix A.1.6) focused on participants' experiences with TextVision compared to traditional tools, their perceptions of its AI-driven features, and areas for improvement. Participants were e.g. asked about the PromptDesigner and prompt recommendation system, and whether the all-in-one design of the interface (document editor, PDF viewer, AI-Assistant) improved efficiency or not. This qualitative approach captured user experiences not fully understood by quantitative measures alone.

8.3. Execution of the User Study

The user study took place from January 6 to February 6, 2025 in the CORE IML in Oldenburg, with each session lasting approximately 90 minutes. The study was conducted to test participants' interactions with TextVision.

Before the study, interested participants were sent an informational document (see Appendix A.1.9) outlining the study's purpose, general information as well as information about data protection. This document allowed participants to familiarize themselves with the study's structure and the tools they would be using in advance. Upon arrival at the CORE IML, participants were asked to fill out an informed consent form (see Appendix A.1.10), confirming their agreement to take part in the study and acknowledging their understanding of the study's procedures.

The sessions were typically scheduled between 12:00 PM - 02:00 PM and 02:30 PM - 04:30 PM on weekdays, with a 30-minute buffer between the two slots to allow for transitions between participants. The study supervisors arrived approximately 30 minutes prior to the first session to set up the necessary equipment, including starting TextVision and uploading a selected PDF⁵⁰ (see Appendix A.1.11) into a new workspace for the participant. This preparation was crucial to ensure that each session ran smoothly in a standardized process and that participants could immediately begin the tasks upon arrival. Additionally, OBS was set up to record the participant's screen throughout the session. This screen recording was used to capture interactions with both Microsoft Word and TextVision, providing additional data for the study's evaluation.

Once a participant arrived, they were greeted and directed to the PC they would later work on. At this point, they filled out the basic questionnaire, providing information about their prior experience with traditional word processors and AI tools. Afterward,

⁵⁰Elitsa. (2024, December 23). How can humans and AI work together to detect deep-fakes? Science Journal for Kids and Teens. <https://www.sciencejournalforkids.org/articles/how-can-humans-and-ai-work-together-to-detect-deepfakes/>

participants received a brief introduction to the study's structure, including a summary of the tasks they would be completing and how long they would have to work on each task. The introduction also included a short tutorial on how to use TextVision to ensure participants had time to feel comfortable with the interface before they started working on the tasks.

The tasks were divided into two parts: Part 1 involved using Microsoft Word, and Part 2 required working with TextVision. This design was intended to provide a direct comparison between a traditional (Microsoft Word) and an AI-powered tool (TextVision). Participants had 30 minutes for each part, totaling one hour of task completion time. During this time, participants were encouraged to ask questions and seek help if needed. The tasks in Part 2 were designed to ensure that participants interacted with all of TextVision's main features, including the document editor, AI assistant, and PDF viewer, so that they are able to give feedback on it. Once the participants completed the tasks, they filled out the NASA-TLX and SUS questionnaires. These questionnaires were followed by the semi-structured interview, where participants were asked to share their thoughts and experiences in more detail. At the end of each session, participants signed a receipt confirming their compensation, and they were given 15€ in cash for their time and effort.

9. Results of the User Study

This section presents the findings from the user study conducted to evaluate the efficiency, usability, and overall user experience of the TextVision interface. The study employed a combination of quantitative and qualitative methods, including standardized questionnaires (SUS & NASA-TLX) as well as a semi-structured interview, to gather comprehensive insights into how participants interacted with TextVision, showcasing the strengths and areas for improvement. The following subsections detail the outcomes of each measurement tool, starting with the Basic Questionnaire, which assessed participants' prior experience with AI tools and word processors.

9.1. Results of the Basic Questionnaire

The Basic Questionnaire (see Appendix A.1.3) had to be filled out by all 19 participants before they interacted with TextVision, aiming to gauge their prior experience with AI tools and traditional word processors. The results revealed a diverse range of familiarity and comfort levels with AI-assisted tools, which allowed participants to be categorized into two distinct groups: low-level and high-level skill users. This classification was primarily based on their responses to Question 5 (see Figure 79), which assessed their comfort level with using AI-assisted tools for document creation. Participants who reported being "very comfortable" or "somewhat comfortable" were classified as high-level skill users, while those who were "neutral", "somewhat uncomfortable" or "very uncomfortable" were categorized as low-level skill users.

Based on this, out of the 19 participants, 6 were identified as high-level skill users (Subject 01, 03, 09, 11, 15 & 16) with a higher familiarity and confidence in using AI tools. These participants reported using AI-powered language models, such as ChatGPT or Microsoft Copilot, more frequently. Several even indicating daily or weekly usage. In contrast, 6 participants were classified as low-level skill users (Subject 06, 08, 13, 17, 18 & 19), with limited experience and comfort in using AI tools. Many in this group had rarely or even never used AI-powered features in word processing tools, and some expressed discomfort with AI-assisted workflows. The remaining 7 participants provided neutral responses. This distribution reflects a balanced representation of skill levels.

The questionnaire also provided insights into participants' proficiency with traditional word processing tools, with the majority rating themselves as intermediate or advanced users. This suggests that while participants were generally adept at using tools like Microsoft Word, their experience with AI-enhanced features varied significantly. The full results of the Basic Questionnaire, including detailed responses to each question, can be found in the Appendix A.1.3. This initial assessment of participants' skill levels was crucial for interpreting their subsequent interactions with TextVision and understanding how prior experience influenced their perceptions of the tool's usability and efficiency.

Subject ID	High Skill (Very comfortable, somewhat comfortable)	Neutral	Low Skill (Somewhat uncomfortable, very uncomfortable)
Subject 01	X		
Subject 02		X	
Subject 03	X		
Subject 04		X	
Subject 05		X	
Subject 06			X
Subject 07		X	
Subject 08			X
Subject 09	X		
Subject 10		X	
Subject 11	X		
Subject 12		X	
Subject 13			X
Subject 14		X	
Subject 15	X		
Subject 16	X		
Subject 17			X
Subject 18			X
Subject 19			X
Total Amount	6	7	6

Figure 79: Grouping based on Question 5 of the Basic Questionnaire

The following sections examine how these skill levels correlated with participant performance and feedback during the study, particularly in terms of cognitive workload and usability. Dividing users into low-level and high-level skill groups will also help illustrate how individuals with varying levels of AI experience perceived TextVision’s features.

9.2. Results of System Usability Scale

The System Usability Scale (see Appendix A.1.4) was filled out by the participants after they completed their tasks using TextVision. The SUS scores provide a quantitative measure of the perceived usability of the system, with higher scores indicating better usability. The average SUS score across all participants was 72.5, which results in the “C+” grade according to a general grading scale (Lewis & Sauro, 2018). Since the median SUS score in the grading scale is 68, TextVision’s score suggests a slightly above-average level of usability. The maximum SUS score recorded was 82.5, achieved by Subject 05, which is classified as a “A” and suggests above-average usability. Conversely, the minimum score

was 60, recorded by Subject 12, which falls into the "D" range, indicating below-average usability. The complete results can be seen in Appendix A.1.7.

The distribution of SUS scores reflects a range of user experiences with TextVision. High-level skill users generally reported higher SUS scores, with several participants in this group scoring above 75, indicating good usability. In contrast, low-level skill users tended to report lower scores, with some falling below 70, suggesting that users with less experience or comfort with AI tools found TextVision less intuitive or user-friendly.

9.3. Results of NASA-TLX

The NASA-TLX questionnaire (see Appendix A.1.5) was used to measure the cognitive workload experienced by participants while using TextVision. The average total NASA-TLX score across all participants was 41.58, with a maximum score of 74 and a minimum score of 17 (see Figure 80). These scores indicate a wide range of cognitive workload experiences among participants. The complete results can be seen in Appendix A.1.8.

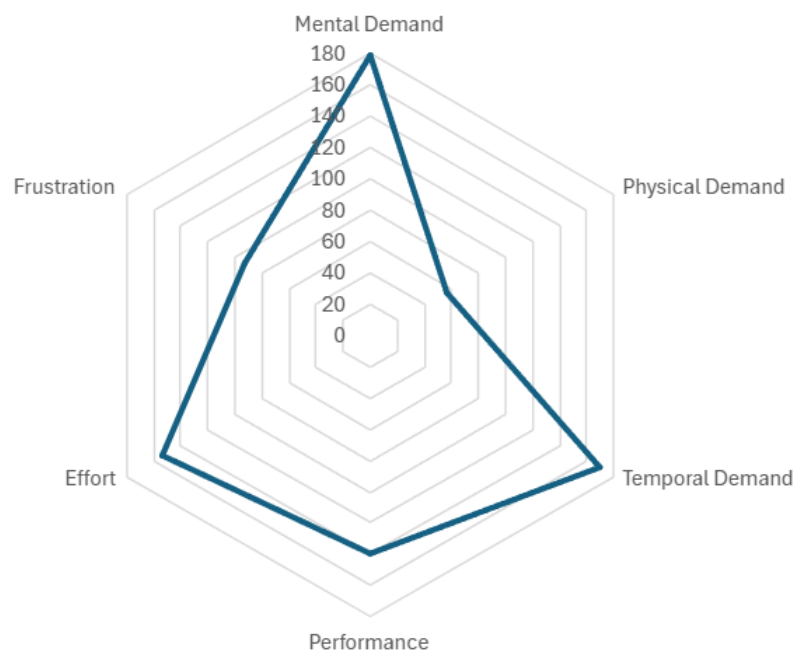


Figure 80: Total Score of each Category

High-level skill users generally reported lower total NASA-TLX scores, suggesting that they experienced less cognitive workload compared to low-level skill users. For example, Subject 01, a high-level skill user, reported a total score of 27, indicating a relatively low

cognitive workload. In contrast, Subject 11, also a high-level skill user, surprisingly reported the highest total score of 74, which indicates a more draining experience with the tool. This variability suggests that even among high-level skill users, individual experiences with TextVision can differ significantly.

Low-level skill users tended to report higher total NASA-TLX scores, reflecting a greater cognitive workload. For instance, Subject 06, a low-level skill user, reported a total score of 66, indicating a high level of mental and physical demand, as well as frustration. Similarly, Subject 18, another low-level skill user, reported a total score of 56, highlighting the challenges faced by users with limited experience or comfort with AI tools.

9.4. Additional Findings

The user study provided important insights into participants' behavior and interaction with TextVision. The following findings summarize the observed patterns of task completion efficiency, need for system prompts, and engagement with multimodal features.

Task Completion Time Participants demonstrated a median task completion time of 23 minutes and 20 seconds. This metric suggests that the platform enables efficient task completion within a reasonable timeframe. In comparison to traditional word processing tools, in this case Microsoft Word, participants took 30 minutes to complete their tasks before time was stopped because the user study had a time frame for completing some tasks. Future studies could further analyze variations in completion times across different user demographics or experience levels to refine TextVision's usability.

Number of prompts A median of three prompts was required for participants to successfully complete their task goals. This finding highlights a moderate level of assistance required, suggesting that while the platform interface is largely intuitive, some tasks may benefit from clearer guidance or additional instructional support, or perhaps this is also influenced by participant experience.

The implications of this finding require further investigation. Potential factors influencing non-engagement could include user preference for traditional input methods, the nature of the assigned tasks that do not require voice interaction, or the need for better feature onboarding.

9.5. Discussion of Interview Feedback & General Insights

The interview feedback from the user study provided valuable insights into participants' experiences with TextVision, revealing both strengths and areas for improvement. The transcripts of the interviews, that were used to write this section, can be found in the Appendix A.2. One of the most frequently mentioned positive aspects was the **Prompt-Designer**, which was praised by multiple participants for its ability to streamline the document creation process. High-level skill users, such as Subject 01 and Subject 15,

found the PromptDesigner particularly useful for refining their inputs and saving time on tasks like summarization. Subject 01 noted: "I didn't have to think as much about phrasing things correctly; the PromptDesigner helped refine my input". Similarly, Subject 15 appreciated the ability to save and reuse prompts, stating: "I liked that you can always go back to the prompts if you have saved one there. I think that would be really cool, especially in the long-term process". This feature was seen as a significant efficiency gain, especially for users who frequently work with similar types of documents.

However, the **Prompt Recommendation** feature received mixed feedback. While some participants found it helpful, others, particularly high-level skill users, felt that the recommendations did not always align with their intent. Subject 11, for example, expressed frustration and said: "I already knew what I wanted, but the recommendations didn't match". Similarly, Subject 06 noted: "The recommendations weren't really in the same context as my chat input". This suggests that while the feature has potential, it may need to be more context-aware or customizable to better meet user needs. Low-level skill users, on the other hand, seemed to benefit more from the recommendations, as they often lacked the confidence or experience to craft their own prompts from scratch. The problems associated with this, have later been addressed and mostly fixed (see Section 9.5.1).

Another recurring positive aspect in the interviews was the **integrated interface** of TextVision, which combines the document editor, PDF viewer, and AI-Assistant into a single screen. Many participants, including Subject 02 and Subject 07, found this integration to be a relevant efficiency gain. Subject 02 remarked: "It definitely made it a lot easier to work quickly, because with the Microsoft Word document I always had to go back and forth between the PDF and the Word document. Here I had everything on one screen instead". This sentiment was shared by Subject 07, who liked the ability to mark text and directly ask questions within the same interface. However, some participants, particularly those in the low-level skill group, found the interface initially overwhelming. Subject 18, for instance, mentioned that it took some time to get used to the layout, but eventually found it intuitive, in the 30 minutes of doing their study tasks.

One notable observation was the lack of engagement with **multimodal features**, such as voice commands. Despite being informed of their availability, almost none of the participants used these features during the study. This suggests that users either found traditional input methods sufficient or were not sufficiently motivated to explore these features. For example, Subject 10 stated: "I didn't need it and didn't feel the need" while Subject 03 mentioned that they generally avoid voice commands due to their perceived lack of accuracy. This indicates that while multimodal features may have potential, they may not be a priority for the current user base. However, it is important to note that the lack of use in this study was likely due to the general preference of the participants. For users with disabilities or those who rely on assistive multimodal technologies, such features are crucial for accessibility. Voice commands and other multimodal interactions could in such cases enhance the usability of the TextVision platform. Therefore, while the

current user base may not have utilized these features, their inclusion is still important to ensure that TextVision is accessible to a broader audience.

In terms of challenges, several participants mentioned difficulties with highlighting text across multiple pages in the PDF viewer. For instance, Subject 13 found it "a bit annoying" to mark text that spanned multiple pages, while Subject 19 noted that they had to mark individual passages separately. This issue was particularly frustrating for users who needed to extract large sections of text, where not everything was on a single page. It is worth noting that this limitation is not unique to TextVision but is a challenge associated with the PDF format in general.

9.5.1. Issues and Bugs identified during the User Study

During the course of the user study, several bugs and issues were identified that impacted the usability and functionality of TextVision. The critical bugs, which were found early on in the user study, were addressed and fixed, with updates merged into the user study branch as soon as they were resolved. Less critical issues, however, were documented and fixed at a later stage. Below are some of the most relevant bugs that were encountered during the study.

One of the critical bugs discovered early on involved the functionality of the PromptDesigner. It was found that the tool was designed to fix all issues regardless of whether they were specifically selected by the user. This led to an unnecessary overhaul of prompts even when the user did not request specific changes, affecting the efficiency. Once identified, this bug was promptly fixed, ensuring that only the selected prompts would be modified in future interactions.

Another notable issue was with the Prompt Recommendations feature. Participants reported that the suggestions provided by the system were often too general and did not align with the direction they intended to take in their documents. This issue arose because the system was using a combination of the context of the uploaded PDF, chat history, and current user input in the chat field to generate the prompts. As a result, the system occasionally produced prompts that were not contextually relevant to the user's needs. To address this, the priority of the user's input was increased in the prompt generation algorithm. This adjustment ensured that the suggestions were more tailored to the user's immediate requirements, improving the relevance and quality of the prompts recommended.

Additionally, a bug was identified where users were only able to use a single prompt created within the PromptDesigner. The combobox on the Home Page, which was intended to allow users to select and apply multiple prompts, didn't function correctly due to an error with the prompt IDs. This issue was resolved by fixing the ID assignment in the code, allowing users to choose and select between multiple prompts as intended.

A visual bug was also discovered during the study, which affected the user interface when interacting with the chat field and PDF Viewer simultaneously. If users marked text within the chat field and crossed the boundary into the PDF Viewer, a popup would

appear showing the chat field a second time. However, due to the complexity of the fix, it couldn't completely be fixed in time and occasionally still resurfaces (see Section 5.11).

9.6. Evaluation of the User Study

This section presents an analysis of the user study conducted to evaluate TextVision based on the previous chapters, focusing on efficiency, usability and cognitive load reduction. This section highlights the research questions and hypotheses that were formulated before the user study was conducted. The study involved 19 of the intended 20 participants and used semi-structured interviews, the System Usability Scale and the NASA-TLX to assess usability and workload.

9.6.1. Research Question

The Research Question aimed to determine if the TextVision interface improves efficiency, usability and overall user experience compared to Microsoft Word. The results of the user study are presented in the following subsections.

Efficiency Gains

The study revealed major efficiency gains when using TextVision:

- Average task completion time using TextVision: 23 minutes and 20 seconds
- Average task completion time with Microsoft Word: 30 minutes
- TextVision was approximately 7 minutes faster, indicating increased efficiency

Usability Ratings

The System Usability Scale scores for TextVision were as follows:

- Average SUS score: 72.5 (usability grade C)
- Highest SUS score: 82.5 (usability grade B)
- Lowest SUS score: 60 (usability grade D)

Here, highly skilled users reported higher SUS scores (>75), indicating a more positive experience.

User Experience

The interviews revealed that many participants found TextVision more intuitive than Microsoft Word. The integrated interface (AI-Assistant, PDF Viewer and Document Editor) was seen as a major efficiency gain. However, some participants struggled with the unfamiliarity and preferred the formatting features of Microsoft Word.

9.6.2. Sub-Question

The Sub-Question Question investigated how the PromptDesigner and Prompt Recommendations reduced cognitive load. The results of the NASA-TLX assessment and interviews are presented in the following subsections.

NASA-TLX Results

- Average NASA-TLX score: 41.58 (moderate workload)
- Lowest score: 17 (low cognitive load)
- Highest score: 74 (high cognitive load)

Here, low-skilled users reported higher cognitive load, indicating difficulty in adapting to AI-assisted workflows.

Interview Insights

Participants reported that PromptDesigner and Prompt Recommendations reduced effort in several ways, including refining inputs and saving time on summarization tasks. However, some challenges were noted, including mismatches between recommendations and user intent, and a preference among experienced users to write prompts manually instead of using Prompt Recommendations.

Interview insights: PromptDesigner & Recommendations Reduced Effort

- Subject 1: "I didn't have to think as much about phrasing things correctly; The PromptDesigner helped refine my input."
- Subject 2: "I didn't have to read the full text before summarizing, which saved effort."
- Subject 10: "I usually type manually, but over time, saved prompts would be much faster."

Challenges with Prompt Recommendations:

- Subject 6: "Recommendations didn't always match my intent."
- Subject 11: "I already knew what I wanted, but recommendations didn't match."
- Some users preferred to write their own prompts for the task that asked them to try out the Prompt Recommendations, and found the recommendations too rigid.

9.6.3. Hypotheses Evaluation

Hypothesis 1: TextVision helps users create documents faster and easier than Microsoft Word

This hypothesis was supported by the evidence. Task completion was faster with TextVision, and users found it easier to create content than to start from a blank page. The built-in AI-Assistant helped reduce effort by automating summaries and writing tasks. However, some users still preferred the familiar Word interface for formatting.

Hypothesis 2: The recommended prompts and PromptDesigner reduce cognitive load compared to traditional tools

This hypothesis was supported in part. NASA-TLX scores showed reduced cognitive load (average: 41.58), and users found AI-Assisted text generation easier than manual writing. However, challenges arose when Prompt Recommendations did not always match user intent, leading to frustration.

Hypothesis 3: The interactive features (AI-Assistant, voice commands) enhance usability compared to traditional tools

This hypothesis was partly supported. The SUS score of 72.5 indicates a C-grade usability. The AI-Assistant was praised for summarizing and organizing information. However, voice commands were not used by any of the participants, suggesting a limited need for them. In addition, UI challenges such as formatting issues and prompt alignment slightly impacted usability.

9.6.4. Design & Development Implications

Based on the results of the User Study, several implications for the design and development of TextVision were identified:

1. Improve Prompt Recommendations by allowing customization and providing better guidance on how to refine prompts.
2. Increase adoption of voice commands through tutorials or incentives.
3. Optimize for low-skilled users by including a guided onboarding experience.

9.7. System Requirements for TextVision

In order to improve the usability and functionality of TextVision based on user feedback, the following system requirements have been formulated with the help of the semi-structured interview, where subjects had to answer the question "In what ways do you think TextVision could be improved to better meet your needs?". Each requirement is categorized according to its importance based on the template of the *MASTeR-Schablonen*⁵¹

⁵¹Sophist. (2019, April 24). Anforderungsschablonen – der MASTeR-Plan für gute Anforderungen. SOPHIST GmbH. <https://blog.sophist.de/2018/03/28/anforderungsschablonen-der-master-plan-fuer-gute-anforderungen/>

(MUST, SHOULD or WILL), the required system functionality and the type of function it represents (automatic system activity, user interaction or interface requirement).

- The system **SHOULD** provide a cleaner and more structured interface as a user interaction to ensure that users can focus on key elements.
- The system **SHOULD** provide clear guidance on the Description field as a user interaction to help new users distinguish it from the Prompt field.
- The system **SHOULD** provide pre-installed prompts as an automatic system activity so that non-technical users can generate responses without creating their own prompts.
- The system **WILL** enable users to efficiently summarize long texts as a user interaction so that experts can quickly extract insights.
- The system **SHOULD** improve prompting capabilities as an automatic system activity so that users can work more efficiently without having to manually refine prompts.
- The system **SHOULD** provide an editing function for chat responses as a user interaction, allowing users to refine outputs within the platform.
- The system **MUST** support dark mode as a user interaction, ensuring a comfortable UI for different lighting conditions.
- The system **SHOULD** generate well-formatted results as an automatic system activity, reducing manual adjustments for users.
- The system **WILL** recommend features based on user behavior as an automatic system activity, helping first-time users navigate efficiently.
- The system **SHOULD** highlight annotation symbols more clearly as a user interaction, improving usability for text annotation tasks.
- The system **MUST** maintain consistent formatting when submitting chat output to the editor as an automatic system activity, preventing users from making manual corrections.
- The system **SHOULD** refine AI-generated responses for conciseness as an automated system activity, eliminating unnecessary phrases.
- The system **SHOULD** improve the accuracy of image analysis as an interface requirement, ensuring reliability when processing visual content.
- The system **WILL** provide a faster method of transferring AI-generated responses to the editor as a user interaction, reducing unnecessary manual deletions.

10. Conclusion

In this documentation the application TextVision, a tool that aims to ease working with PDF documents with the help of an integrated AI assistant, was described. As mentioned, before the actual implementation, two research questions were formulated. These research questions could ultimately be answered through the user study that was carried out. The first research question and the derived hypothesis, whether TextVision actually improves the efficiency, usability and overall user experience of the document creation process in comparison to traditional word processing tools, was ultimately confirmed. Regarding the usability, the user study yielded an average SUS score of 72.5, with anything above 68 being considered above-average. The application is thus overall usable slightly beyond an acceptable level, having a C+ SUS rating. As expected, users that previously classified themselves as high-skill scored higher SUS scores than low-skill users, though even low-skill users score slightly higher than 68. The results regarding the workload-measuring NASA-TLX were 41.58 on average, suggesting a moderate cognitive load. Expectedly, low-skill users reported a higher workload than high-skill users. It was also measured that on average, the task completion with TextVision took 23.33 minutes, while it took 30 minutes on average with Microsoft Word. Task completion with TextVision therefore yielded a reduced task time of roughly 7 minutes, which is 22.23 % in comparison. Hence, according to the results, it is suggested that TextVision does improve efficiency, usability and overall user experience. The underlying sub-question in what ways the PromptDesigner and prompt recommendations features reduce the cognitive load during the document creation process could also be answered. A median of 3 used prompts per task indicates a moderate level of needed assistance. Some users found the prompt functionalities intuitive while others needed additional support. Prompt-related features ultimately received mixed feedback, with high-skilled users considering some prompt recommendations misaligned with their intent and low-skilled users benefiting from them. The PromptDesigner itself was generally regarded as useful and timesaving by the participants.

Though the overall perception of TextVision was generally positive, the application is not without issues, which sometimes lie on a technical level, though they are not necessarily noticeable by the user in every case. One is the potential longer processing time, especially when larger amounts of data are involved or when a voice prompt first needs to be transcribed via Whisper before it can be processed by the selected LLM. Hence, there may be scenarios where the direct OpenAI API would be faster. Also, due to the nature of LLMs, it can happen that identical prompts are rated slightly different. Though TextVision implements some solutions that aim to mitigate the problem of hallucinations in generated content, they cannot be fully prevented, potentially leading to user frustration depending on the task. The system also lacks an automatic shortening of the context window, leading to users potentially exceeding the model's token limit, depending on the length of their input. Another limitation are incomplete text extractions when working large PDF files that exceed 50 MB.

The implementation of TextVision took roughly six months. In the meantime, the world of AI and LLMs was expanding rapidly with new technologies and solutions to existing problems emerging along the way. Thus, it is possible that the novelties that TextVision aims to implement are or will be covered by other applications as well.

The goal of TextVision was to bridge a gap in the field of human AI interaction and the work with research documents, which was primarily realized by combining several prompting techniques in order to create a more efficient workflow. Given the results of the user study, especially the predominantly positive qualitative feedback regarding the PromptDesigner and the general functionality of the tool, this overall goal can thus ultimately be considered successful. Naturally, the application still has room for future implementation and adjustments, which are discussed in the following chapter.

11. Outlook: Development and Expansion

Due to the limited time in the project, some aspects of the project have been placed in the future. This chapter discusses some aspects that could be implemented with more time or in further research groups.

- **Expansion of LLM Integration:** One of the most important aspects to keep optimizing is the implementation of new LLM-models as they are released. Different models offer different strengths in reasoning, summarization and domain-specific knowledge. By combining multiple LLMs the system could improve in aspects of accuracy and adaptability. Specifically, implementing a model orchestration layer that selectively routes queries to specialized models based on task requirements would optimize performance across different use cases.
- **Multi-Model Verification Chains:** Implementing verification pipelines that connect multiple models in sequence would improve output reliability. This approach would use specialized models to verify factual claims, check logical consistency, and ensure compliance with specified constraints. The system would flag any discrepancies for user review, enhancing the overall accuracy of generated content.
- **Simplified System Prompt Management:** The current approach to system prompt configuration requires code-level modifications. Developing a user-friendly interface for system prompt customization would improve maintainability and allow for more rapid adaptation to different use cases. This could include a template-based approach with modular components that can be assembled through a visual interface.
- **Advanced PDF Processing and Generation:** The extension of the capabilities of LLMs necessitates an enhancement of their interaction with diverse data formats, particularly PDFs. TextVision has a primary focus on working with documents, therefore basic tools and functions were implemented for specific use cases. With more time, TextVision could be expanded upon to cover more use cases and offer more functionalities to the user. One concrete improvement would be AI-driven extraction and processing of PDF content, such as table extraction, which remains a significant challenge for current models.
- **Version Control and Optimization for Prompts:** A key aspect of TextVision is its prompting capability. To further improve the optimization of LLM responses, a structured versioning system could be helpful to the user. This would allow comparative analysis of different formulations and enable continuous improvement of AI-generated output by identifying the most effective prompt structure for different tasks. The storage and analysis of prompt histories could provide a basis for further optimization.
- **Voice Control and Interactive Voice Bots:** In order to facilitate hands-free AI interaction, it is necessary to implement an advanced voice interface. TextVision

provides basic speech-to-text and text-to-speech functionality, which could be further developed. The system could allow seamless interaction with LLM-based assistants. The goal could be to implement a real-time conversation workflow that acts like a real assistant for the user, helping to get work done faster in a multitasking environment.

- **Enhanced Source Attribution and Transparency:** Current implementation only displays the source document for information, without precise attribution of specific content elements. A more detailed attribution system could be developed that highlights exactly which sections of source documents contributed to each part of the generated output. This would improve transparency and trustworthiness, particularly for academic and research applications where precise citation is essential.
- **Efficient Token Management:** Current implementation does not optimize token usage. Developing preprocessing mechanisms that filter non-relevant content before LLM processing would allow for more efficient use of context windows and reduce operational costs. Techniques such as semantic chunking and relevance scoring could ensure only the most pertinent information is included in prompts.
- **Processing Speed Optimization:** Response time optimization is particularly crucial for the recommendation feature, where users expect near real-time suggestions. Implementing OpenAI's newer real-time API could significantly reduce response latency. Additionally, optimizing the preprocessing pipeline for PDFs, vector store creation, and assistant initialization would improve overall system responsiveness.
- **Framework Research and Prompt Engineering:** A further improvement would be to add more future state-of-the-art prompting frameworks that users can use to improve their work. In addition, these new techniques could be used to further develop TextVision's prompting concepts in the technical architecture of TextVision. These techniques could enhance the prompting capabilities and therefore the results.
- **Access to more research knowledge:** A substantial enhancement to the AI-generated output would be the expansion of the knowledge base of the connected LLM models. One idea that has emerged is the integration of a scientific database or open-access repository. The incorporation of additional knowledge input could serve to enhance the quality of chat responses and facilitate greater accessibility to complex academic content.

The continuous advancement of LLM technology presents both opportunities and challenges for maintaining TextVision as a current system. Future development should prioritize features that deliver measurable benefits to user's workflows rather than implementing technology for its own sake. Particularly promising areas include response time optimization, enhanced source attribution, efficient token management, and verification mechanisms that leverage multiple specialized models. As the underlying AI capabilities evolve, TextVision can transition from a document analysis tool to a comprehensive research assistant that adapts to individual workflows across multiple interaction modalities.

Bibliography

- Agarwal, E., Singh, J., Dani, V., Magazine, R., Ganu, T., & Nambi, A. (2024). PromptWizard: Task-Aware Prompt Optimization Framework. *arXiv preprint arXiv:2405.18369*.
- Alpaydin, E. (2022). *Maschinelles Lernen*. De Gruyter Oldenbourg. <https://doi.org/10.1515/9783110740196>
- Amaratunga, T. (2023). *Understanding Large Language Models: Learning Their Underlying Concepts and Technologies*. Apress. <https://doi.org/10.1007/979-8-8688-0017-7>
- Amatriain, X. (2024, January). Prompt Design and Engineering: Introduction and Advanced Methods. <https://arxiv.org/abs/2401.14423>
- Anshari, A. (2024). A Systematic Study of Multimedia Implementation and its Impact towards User Engagement. *Journal of Educators Online*, 21(2), n2.
- Asemi, A., Asemi, A., & Tahaei, H. (2022). Human-centred design for interactive systems. <https://doi.org/10.1108/lht-02-2022-0091>
- Balzert, H. (2011). *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Spektrum Akademischer Verlag. <https://doi.org/10.1007/978-3-8274-2246-0>
- Battle, R., & Benson, E. (2008). Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semantics: Science, Services and Agents on the World Wide Web*, 6, 61–69. <https://doi.org/10.1016/j.websem.2007.11.002>
- Baum, C., Bengel, G., Kunze, M., & Stucky, K.-U. (2015). Client-Server-Modell und Web-Services. In *Masterkurs Parallele und Verteilte Systeme: Grundlagen und Programmierung von Multicore-Prozessoren, Multiprozessoren, Cluster, Grid und Cloud* (pp. 107–127). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-8348-2151-5_3
- Bhattacharyya, S., Snasel, V., Hassanien, A. E., Saha, S., & Tripathy, B. K. (2020). *Deep Learning: Research and Applications*. De Gruyter. <https://doi.org/10.1515/9783110670905>
- BITKOM. (2017). *Künstliche Intelligenz: wirtschaftliche Bedeutung, gesellschaftliche Herausforderung, menschliche Verantwortung*. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien. <https://books.google.de/books?id=8CIwzgEACAAJ>
- Bourhis, P., Reutter, J. L., & Vrgoč, D. (2020). JSON: Data model and query languages. *Information Systems*, 89, 101478. <https://doi.org/10.1016/j.is.2019.101478>
- Bsharat, S. M., Myrzakhan, A., & Shen, Z. (2023). Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4. <https://doi.org/10.48550/arxiv.2312.16171>
- Buxmann, P., & Schmidt, H. (2021). *Grundlagen der Künstlichen Intelligenz und des Maschinellen Lernens*. Springer Gabler.

- Caine, K. (2016). Local Standards for Sample Size at CHI. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 981–992. <https://doi.org/10.1145/2858036.2858498>
- Casheekar, A., Lahiri, A., Rath, K., Prabhakar, K. S., & Srinivasan, K. (2024). A contemporary review on chatbots, AI-powered virtual conversational agents, ChatGPT: Applications, open challenges and future research directions. *Computer Science Review*, 52, 100632.
- Clark, J. A. (2020). Application Programming Interface. *The SAGE International Encyclopedia of Mass Media and Society*. <https://api.semanticscholar.org/CorpusID:33056776>
- Dahm, M. H., & Zehnder, V. (2023). Grundlagen der KI. In *Moderne Personalführung mit Künstlicher Intelligenz* (pp. 3–16). Springer Gabler. https://doi.org/10.1007/978-3-658-43138-9_2
- Desmond, M., Ashktorab, Z., Pan, Q., Dugan, C., & Johnson, J. M. (2024). EvalLLM: LLM assisted evaluation of generative outputs. *Companion Proceedings of the 29th International Conference on Intelligent User Interfaces*, 30–32. <https://doi.org/10.1145/3640544.3645216>
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient Finetuning of Quantized LLMs. <https://arxiv.org/abs/2305.14314>
- Document management — Portable document format — Part 1: PDF 1.7*. (2008). International Organization for Standardization. <https://www.iso.org/standard/51502.html>
- Erik. (2024). Was sind Large Multimodal Models (LMM)? *Healthcare Digital*. Retrieved March 1, 2024, from <https://www.healthcare-digital.de/was-sind-large-multimodal-models-lmm-a-b6fb26c21ec883cc2355688c1664bdaa/>
- Fagbohun, O., Harrison, R. M., & Dereventsov, A. (2024, February). An Empirical Categorization of Prompting Techniques for Large Language Models: A Practitioner’s Guide. <https://arxiv.org/abs/2402.14837>
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext transfer protocol-HTTP/1.1* (tech. rep.).
- Figuerola-Torres, M. (2025). The Three Social Dimensions of Chatbot Technology. *Philosophy & Technology*, 38(1), 1–23.
- Fok, R., Kambhamettu, H., Soldaini, L., Bragg, J., Lo, K., Hearst, M. A., Head, A., & Weld, D. S. (2023). Scim: Intelligent Skimming Support for Scientific Papers. *Proceedings of the 28th International Conference on Intelligent User Interfaces, IUI 2023, Sydney, NSW, Australia, March 27-31, 2023*, 476–490. <https://doi.org/10.1145/3581641.3584034>
- Giessler, P., Gebhart, M., Sarancin, D., Steinegger, R., & Abeck, S. (2015). Best practices for the design of restful web services. *International conferences of software advances (ICSEA)*, 392–397.
- Google. (2024). *API – Übersicht*. Retrieved March 12, 2024, from <https://ai.google.dev/docs/gemini-api-overview?hl=de>

- Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., & et al., A. K. (2024). The Llama 3 Herd of Models. <https://arxiv.org/abs/2407.21783>
- Gunawan, R., & Rahmatulloh, A. (2019). JSON Web Token (JWT) untuk Authentication pada Interoperabilitas Arsitektur berbasis RESTful Web Service [JSON Web Token (JWT) for Authentication in RESTful Web Service-based Architecture Interoperability]. *Jurnal Edukasi dan Penelitian Informatika (JEPIN)*, 5, 74. <https://doi.org/10.26418/jp.v5i1.27232>
- Gusenbauer, M., & Haddaway, N. R. (2021). What every researcher should know about searching—clarified concepts, search advice, and an agenda to improve finding in academia. *Research synthesis methods*, 12(2), 136–147.
- Han, Z., Gao, C., Liu, J., Zhang, J., & Zhang, S. Q. (2024). Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey. <https://arxiv.org/abs/2403.14608>
- Harwardt, M., & Köhler, M. (2023). *Künstliche Intelligenz entlang der Customer Journey*. Springer Gabler.
- Hein, L., Högemann, M., Illgen, K.-M., Stattkus, D., Kochon, E., Reibold, M.-G., Eckle, J., Seiwert, L., Beinke, J. H., Knopf, J., et al. (2024). ChatGPT als Unterstützung von Lehrkräften—Einordnung, Analyse und Anwendungsbeispiele. *HMD Praxis der Wirtschaftsinformatik*, 61(2), 449–470.
- Hewett, T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., & Verplank, W. (1992). *ACM SIGCHI Curricula for Human-Computer Interaction*. <https://doi.org/10.1145/2594128>
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. <https://arxiv.org/abs/2106.09685>
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023). Mistral 7B. <https://arxiv.org/abs/2310.06825>
- Jiang, E., Olson, K., Toh, E., Molina, A., Donsbach, A., Terry, M., & Cai, C. J. (2022). Promptmaker: Prompt-based prototyping with large language models. *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 1–8.
- Jin, H., Han, X., Yang, J., Jiang, Z., Liu, Z., Chang, C.-Y., Chen, H., & Hu, X. (2024). LLM Maybe LongLM: Self-Extend LLM Context Window Without Tuning. <https://arxiv.org/abs/2401.01325>
- Jones, M., Bradley, J., & Sakimura, N. (2015). *RFC 7519: JSON Web Token (JWT)*.
- Jungmann, F., Kühn, S., & Kämpgen, B. (2018). *Grundlagen und Einsatzmöglichkeiten von Natural Language Processing (NLP) in der Radiologie*. Springer Medizin Verlag. <https://doi.org/10.1007/s00117-018-0426-0>
- Khan, M., Tran, P.-N., Pham, N. T., El Saddik, A., & Othmani, A. (2025). MemoCMT: Multimodal Emotion Recognition Using Cross-Modal Transformer-Based Feature Fusion [Received: 26 June 2024; Accepted: 04 February 2025; Published: 14 Febru-

- ary 2025]. *Scientific Reports*, 15, 5473. <https://doi.org/10.1038/s41598-025-89202-x>
- Khan, N., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Mahmoud Ali, W. K., Alam, M., Shiraz, M., & Gani, A. (2014). Big Data: Survey, Technologies, Opportunities, and Challenges. *The Scientific World Journal*, 2014(1), 712826. <https://doi.org/https://doi.org/10.1155/2014/712826>
- Kim, J., Flanagan, R. C., Haviland, N. E., Sun, Z., Yakubu, S. N., Maru, E. A., & Arnold, K. C. (2024). Towards Full Authorship with AI: Supporting Revision with AI-Generated Views (A. Soto & E. Zangerle, Eds.). 3660. <https://ceur-ws.org/Vol-3660/paper17.pdf>
- Kim, Y., Lee, J., Kim, S., Park, J., & Kim, J. (2024). Understanding Users' Dissatisfaction with ChatGPT Responses: Types, Resolving Tactics, and the Effect of Knowledge Level. *Proceedings of the 29th International Conference on Intelligent User Interfaces, IUI 2024, Greenville, SC, USA, March 18-21, 2024*, 385–404. <https://doi.org/10.1145/3640543.3645148>
- Kirste, M., & Schürholz, M. (2019). Einleitung: Entwicklungswege zur KI. In *Künstliche Intelligenz*. Springer Vieweg. https://doi.org/10.1007/978-3-662-58042-4_1
- Knoth, N., Tolzin, A., Janson, A., & Leimeister, J. M. (2024). AI literacy and its implications for prompt engineering strategies. *Computers and Education: Artificial Intelligence*, 6, 100225. <https://doi.org/10.1016/j.caeai.2024.100225>
- Lewis, J., & Sauro, J. (2018). Item Benchmarks for the System Usability Scale. 13, 158–167.
- Lin, C.-Y. (2004). ROUGE: A Package for Automatic Evaluation of Summaries. *Text Summarization Branches Out*, 74–81. <https://aclanthology.org/W04-1013/>
- Lin, G., Feng, T., Han, P., Liu, G., & You, J. (2024, November). Arxiv Copilot: A Self-Evolving and Efficient LLM System for Personalized Academic Assistance. In D. I. Hernandez Farias, T. Hope, & M. Li (Eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 122–130). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2024.emnlp-demo.13>
- Lin, X., Dai, Z., Verma, A., Ng, S.-K., Jaillet, P., & Low, B. K. H. (2024). Prompt optimization with human feedback. *arXiv preprint arXiv:2405.17346*.
- Luber, S. (2023). Was ist ein Large Language Model? *BigData-Insider*. Retrieved February 20, 2024, from <https://www.bigdata-insider.de/was-ist-ein-large-language-model-a-d735d93bbc24d3c4091de8ce25aa36e8/?print>
- Masson, D., Malacria, S., Casiez, G., & Vogel, D. (2024). Directgpt: A direct manipulation interface to interact with large language models. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–16.
- Mathur, P., Siu, A., Manjunatha, V., & Sun, T. (2024, August). DocPilot: Copilot for Automating PDF Edit Workflows in Documents. In Y. Cao, Y. Feng, & D. Xiong (Eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)* (pp. 232–246). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2024.acl-demos.22>

- Melnikov, A., & Fette, I. (2011, December). The WebSocket Protocol. <https://doi.org/10.17487/RFC6455>
- Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large Language Models: A Survey. <https://arxiv.org/abs/2402.06196>
- Mozilla Developer Network. (2024). Web Speech API Documentation [Accessed: 2024-02-27]. https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API
- Mozilla Developer Network. (2025). MediaRecorder [Accessed: 2025-02-28]. <https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder>
- Neshaei, S. P., Rietsche, R., Su, X., & Wambsganss, T. (2024). Enhancing Peer Review with AI-Powered Suggestion Generation Assistance: Investigating the Design Dynamics. *Proceedings of the 29th International Conference on Intelligent User Interfaces*, 88–102. <https://doi.org/10.1145/3640543.3645169>
- Olan, M. (2003). Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, 19(2), 319–328.
- Oluwatosin, H. S. (2014). Client-server model. *IOSR Journal of Computer Engineering*, 16(1), 67–71.
- Oviatt, S., Schuller, B., Cohen, P. R., Sonntag, D., Potamianos, G., & Krüger, A. (Eds.). (2017). *The Handbook of Multimodal-Multisensor Interfaces: Foundations, User Modeling, and Common Modality Combinations - Volume 1* (Vol. 1). Association for Computing Machinery; Morgan & Claypool.
- Petridis, S., Wedin, B. D., Wexler, J., Pushkarna, M., Donsbach, A., Goyal, N., Cai, C. J., & Terry, M. (2024). ConstitutionMaker: Interactively Critiquing Large Language Models by Converting Feedback into Principles. *Proceedings of the 29th International Conference on Intelligent User Interfaces, IUI 2024, Greenville, SC, USA, March 18-21, 2024*, 853–868. <https://doi.org/10.1145/3640543.3645144>
- Plank, B. (2023). *ChatGPT is the start of a new NLP era – but there is still a long way to go*. Retrieved March 12, 2024, from <https://www.lmu.de/en/newsroom/news-overview/news/chatgpt-is-the-start-of-a-new-nlp-era---but-there-is-still-a-long-way-to-go.html>
- Qigang, L., & Sun, X. (2012). Research of Web Real-Time Communication Based on Web Socket. *International Journal of Communications, Network and System Sciences*, 05, 797–801. <https://doi.org/10.4236/ijcns.2012.512083>
- Rodriguez, A. (2008). Restful web services: The basics. *IBM developerWorks*, 33(2008), 18.
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386–408. <https://doi.org/10.1037/h0042519>
- Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024, February). A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. <https://arxiv.org/abs/2402.07927>
- Sani, L., Iacob, A., Cao, Z., Marino, B., Gao, Y., Paulik, T., Zhao, W., Shen, W. F., Aleksandrov, P., Qiu, X., & Lane, N. D. (2024). The Future of Large Language Model Pre-training is Federated. <https://arxiv.org/abs/2405.10853>

- Shang, Y., & Fu, T. (2024). Multimodal fusion: A study on speech-text emotion recognition with the integration of deep learning. *Intelligent Systems with Applications*, 24, 200436.
- Sheng, Z., Du, Z., Lu, H., Zhang, S., & Ling, Z.-H. (2025). Unispeaker: A Unified Approach for Multimodality-driven Speaker Generation. *arXiv preprint arXiv:2501.06394*.
- Singh, N., Wang, L. L., & Bragg, J. (2024). FigurA11y: AI Assistance for Writing Scientific Alt Text. *Proceedings of the 29th International Conference on Intelligent User Interfaces*, 886–906. <https://doi.org/10.1145/3640543.3645212>
- Skvorc, D., Horvat, M., & Srbljic, S. (2014). Performance evaluation of WebSocket protocol for implementation of full-duplex web streams. *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1003–1008. <https://doi.org/10.1109/MIPRO.2014.6859715>
- Sonnet, D. (2022). *Neuronale Netze kompakt, Vom Perceptron zum Deep Learning*. Springer Vieweg.
- Sučik, S., Skala, D., Švec, A., Hraška, P., & Šuppa, M. (2023). Prompterator: Iterate efficiently towards more effective prompts. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 471–478.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Vrbančič, G., & Podgorelec, V. (2020). Transfer Learning With Adaptive Fine-Tuning. *IEEE Access*, 8, 196197–196211. <https://doi.org/10.1109/ACCESS.2020.3034343>
- Wagenknecht, C. (2004). Verteilte Anwendungen. In *Programmierparadigmen: Eine Einführung auf der Grundlage von Scheme* (pp. 117–140). Vieweg+Teubner Verlag. https://doi.org/10.1007/978-3-322-80082-4_4
- WebKit. (2025). WebKit Documentation [Accessed: 2025-02-28]. <https://docs.webkit.org/>
- Wu, J., Gan, W., Chen, Z., Wan, S., & Yu, P. S. (2023). Multimodal Large Language Models: A Survey.
- Wu, S., Shen, H., Weld, D. S., Heer, J., & Ribeiro, M. T. (2023). ScatterShot: Interactive In-context Example Curation for Text Transformation. *Proceedings of the 28th International Conference on Intelligent User Interfaces, IUI 2023, Sydney, NSW, Australia, March 27-31, 2023*, 353–367. <https://doi.org/10.1145/3581641.3584059>
- Yang, Y., Tao, C., & Fan, X. (2024). LoRA-LiteE: A Computationally Efficient Framework for Chatbot Preference-Tuning. *arXiv preprint arXiv:2411.09947*.
- Zamfirescu-Pereira, J., Wong, R. Y., Hartmann, B., & Yang, Q. (2023). Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 1–21.

A. Appendix

A.1. User Study

A.1.1. Application for Ethical Approval

Self-Evaluation for Ethics – Fast-Track Process

- The following form is a self-evaluation process for research studies.
- If all questions on the provided checklist are self-assessed as innocuous (answered with NO), you may proceed with your study.
- **Please consult** the information provided on Page 5 and 6 carefully. **You are responsible for adhering to legal and ethical regulations.**
- **Answering truthfully falls under your responsibility! It is your obligation to provide all necessary information!**
- Refer to the informed consent generator¹ to create a correct and GDPR-compliant informed consent form for your study. You must also provide the consent form you intend to use for the study.

Procedure:

1. The form consists of two sections: I. General Information, and II. Fast-Track Checklist. Please complete all required parts. Please, answer truthfully and provide all relevant information. The Ethics Committee reserves the right to review forms submitted on a random basis.
2. Send the completed form and your consent form to ethik-sek@dfki.de. Please indicate if you still require an official ethics vote by the committee².

IF you answered all question in part II with “NO”:
you may immediately proceed with your study.

ELSE
you are required to submit a full application³ for review by the ethics committee.

¹ Available as a web form here <https://projects.dfki.uni-kl.de/dws/informed-consent-generator/>.

² E.g., if requested by your funding agency. Be prepared for a turn-around time of approx. one to two months.

³ A full application form is not yet available. Please refer to Page 5 for the procedure or contact ethik-sek@dfki.de in this case.

I. General Information

The context of the study or series of studies. You may select more than one:

- | | |
|------------------------------------------------|-----------------------------------------------------------|
| <input type="checkbox"/> Internship or project | <input type="checkbox"/> Industry funded research project |
| <input type="checkbox"/> Bachelor thesis | <input type="checkbox"/> Publicly funded research project |
| <input type="checkbox"/> Master thesis | <input type="checkbox"/> Other (please specify) |
| <input type="checkbox"/> PhD project | _____ |

Responsible experimenter(s) (The one(s) conducting the study):

Name, given name(s): _____

Research group(s): _____

Position(s): _____

Email address(es): _____

Responsible Supervisors (PI, Project Manager, Thesis supervisor, etc.):

Name, given name(s): _____

Research group(s): _____

Position(s): _____

Email address(es): _____

Other participating institutions (if applicable).

Affiliation to other studies:

Is this a study within the framework of a project or another study for which a vote of the ethical review board is already available or is the current planned study designed analogue to a study for which an approval of the ethical review board is already available?

- ☐ no (continue with checklist)
- ☐ yes

If yes, please indicate the abbreviated designation of the study:

Name of the research project:

Name and brief information on the objectives of the study:


[illegible][illegible]

II. Fast-Track Checklist

	YES	NO
1. Will the study involve individuals who belong to a vulnerable group or who may not be able to provide their own consent to participate such as: children and adolescents under the age of 18, people with learning disabilities, senior citizens, residents of a hospital or nursing home, or people with physical or cognitive disabilities?		
2. Does the study utilize covert observation, deceiving, or any other method that does not ensure full disclosure to participants, or informed consent?		
3. Is it necessary actively to mislead people concerning the purpose of the study?		
4. Will the participants be subject to any invasive or potentially harmful procedures?		
5. Is there a risk that the study will cause psychological distress, fear, fatigue, or other negative effects in participants beyond what would typically be expected in everyday life?		
6. Is there a risk that the study will cause physical distress, pain, discomfort, or more beyond what would typically be expected in everyday life by the participants?		
7. Is it expected that participants are going to suffer from physiological stress, anxiety, exhaustion, physical pain, or other negative effects after the research procedure?		
8. Will the study involve administering drugs, placebos, or other substances (such as foods, beverages, vitamin supplements) to participants, or will participants undergo any invasive or potentially harmful procedures?		
9. Is it necessary to ask questions on subjects of an intimate nature for the respondents or the answer to which could be conceived as stigmatizing (e.g., relating to illegal or deviant behavior)?		
10. Will the research involve the use of sensitive data? (genetic/health/biometric data, ethnicity, political opinions, religious or philosophical beliefs, trade union membership, etc.)		
11. Will the project involve data that could potentially be used to obtain information about sensitive content (such as those listed in question 10) without the prior consent of the participants?		

I confirm that all information in this application is correct and accurate to the best of my knowledge.

Place, date


Signature of the executive researcher

Fasttrack Procedure

The Fast Track procedure is designed for research projects which do not require an extensive review by the DFKI Ethics Board (e.g., questionnaire studies without sensitive questions, short studies with office-like activities, etc.).

This procedure can be used to confirm a basic ethical clearance through the information provided. The prerequisite for the basic ethical clearance will be an unanimously “no” answer to all the questions of the Checklist. If this is applicable to your case send us only the filled and signed form, including your generated informed consent form. You may immediately proceed with your study. Please note that the Ethics Committee reserves the right to review forms submitted on a random basis.

If one or more of the Checklist questions have been answered with “yes”, please provide detailed reasoning on each individual checklist question answered with “yes” on how you plan to mitigate potential risk and/or ethical concerns. It is your obligation to address the rationale for and necessity of the points that were answered with “yes,” and describe how you will ensure compliance with ethics guidelines. You may use the form in the appendix (Page 7) for this. Send all documents (this fast-track form, your detailed explanation, informed consent form) to ethik-sek@dfki.de and wait for the decision of the DFKI Ethics Board before initiating a research procedure.

If you have any additional inquiries, we encourage you to reach out to the ethics board for clarification.

Legal Regulations and Guidelines for Scientific Work

By submitting the application, you confirm that:

- you are aware of and have taken into consideration the applicable legal regulations and guidelines pertaining to your research project in their current versions. This includes, but is not limited to:
 - o the legal provisions related to data protection as outlined in the Federal Data Protection Act (BDSG),
 - o the (EU) 2016/679 Regulation of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the Processing of Personal Data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation),
 - o the ethical guidelines of the ACM (Code of Ethics and Professional Conduct), the Association of Internet Researchers, the German Informatics Society, the German Psychological Society, and the DFG (Guidelines of Good Scientific Work).
- all information provided in the form and, if applicable, in additional documents is correct to the best of your knowledge.
- in the case of student work, you further confirm that the students have been informed about the legal regulations and ethical guidelines.

Expanded information form

To be completed only if one or more questions were answered with “yes” in the Checklist. You may provide additional information if you deem necessary. Try to be as transparent as possible to allow for an adequate review of your research procedure.

Explicitly comment on the rationale behind any question answered with “yes”. State why it necessary to reach your research objective.

Explicitly state how you plan to mitigate any potential ethical concerns and/or related risks related to these.

A.1.2. Promotional Text

Hello everyone!

Participants wanted for innovative user study: TextVision - The future of document creation!

Are you ready to experience the next generation of word processing? The AIM project group at the University of Oldenburg in cooperation with the DFKI is looking for participants for an exciting user study!

What can you expect?

- Test our revolutionary AI-supported platform for document creation
- Experience innovative features such as the "PromptDesigner" and AI Assistant

Details:

- Period: January 6th - February 6th, 2025 (exact date by arrangement)
- Duration: approx. 90 minutes
- Location: CORE IML, Oldenburg
- Remuneration: €15 per participation

Your task:

You create a document using our TextVision interface and test various functions such as the AI Assistant, the PromptDesigner and voice commands. To do this, you will complete various tasks given by us.

Requirements:

- Minimum age: 18 years
- No previous knowledge required - people without technical experience are also welcome!

Data protection:

We record the screen during processing and conduct a short interview (audio recording only). All data is treated confidentially, anonymized and used exclusively to evaluate our prototype.

Interested?

Contact us by email at jannik.podszun@uni-oldenburg.de or melis.aslan@uni-oldenburg.de. Further information and appointments will be made by email.

Shape the future of word processing with us - we look forward to your participation!

A.1.3. Basic Questionnaire

Questionnaire

1. How often do you use word processing tools (e.g., Microsoft Word, Google Docs)?

- ☐ Daily
- ☐ Several times a week
- ☐ Once a week
- ☐ A few times a month
- ☐ Rarely or never

2. How would you rate your proficiency with word processing tools?

- ☐ Beginner
- ☐ Intermediate
- ☐ Advanced
- ☐ Expert

3. Have you used AI-powered language models (e.g., ChatGPT, MS Copilot) before?

- ☐ Yes
- ☐ No

4. If yes, how frequently do you use AI language models?

- ☐ Daily
- ☐ Several times a week
- ☐ Once a week
- ☐ A few times a month
- ☐ Rarely

5. How comfortable are you with using AI-assisted tools for document creation?

- ☐ Very comfortable
- ☐ Somewhat comfortable
- ☐ Neutral
- ☐ Somewhat uncomfortable
- ☐ Very uncomfortable

6. Have you ever used AI-powered features in word processing tools (e.g., Microsoft Editor, Grammarly)?

- ☐ Yes
- ☐ No

7. How important do you think AI assistance is for document creation?

- ☐ Very important
- ☐ Somewhat important
- ☐ Neutral
- ☐ Not very important
- ☐ Not at all important

A.1.4. System Usability Scale (SUS)

System Usability Scale (SUS)

Strongly Disagree

Strongly Agree

I think that I would like to use TextVision frequently.

1	2	3	4	5

I found TextVision unnecessarily complex.

1	2	3	4	5

I thought TextVision was easy to use.

1	2	3	4	5

I think that I would need the support of a technical person to be able to use TextVision.

1	2	3	4	5

I found the various functions in TextVision were well integrated.

1	2	3	4	5

I thought there was too much inconsistency in TextVision.

1	2	3	4	5

I would imagine that most people would learn to use TextVision very quickly.

1	2	3	4	5

I found TextVision very awkward to use.

1	2	3	4	5

I felt very confident using TextVision.

1	2	3	4	5

I needed to learn a lot of things before I could get going with TextVision.

1	2	3	4	5

A.1.5. NASA-TLX

A.1.6. Interview Questions

Semi-structured interview

1. How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

2. What aspects of TextVision's interface did you find most helpful for document creation?

3. How did the PromptDesigner and prompt recommendation features impact your workflow?

4. Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

5. Can you describe any challenges you encountered while using TextVision?

6. How did the AI-Assistant's capabilities compare to your expectations?

7. Did you use the voice command feature? If so, how did it affect your document creation process?

8. In what ways do you think TextVision could be improved to better suit your needs?

9. How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

10. Would you consider using TextVision for your future document creation tasks? Why or why not?

A.1.7. Results: System Usability Scale (SUS)

System Usability Scale (SUS)

Subject ID	Question 01	Question 02	Question 03	Question 04	Question 05	Question 06	Question 07	Question 08	Question 09	Question 10	Total Score	SUS Score
1	4	1	4	2	4	1	4	1	5	1	27	67,5
2	4	1	4	2	4	1	5	1	3	1	26	65
3	5	1	3	3	5	2	3	1	3	4	30	75
4	3	1	5	1	4	1	4	1	4	2	26	65
5	4	2	5	5	4	2	4	1	3	3	33	82,5
6	3	1	5	1	4	2	5	2	4	3	30	75
7	5	1	5	2	5	1	5	1	5	1	31	77,5
8	4	1	4	3	5	2	5	2	3	2	31	77,5
9	4	2	4	1	5	2	5	1	4	3	31	77,5
10	5	2	5	1	4	2	5	1	5	2	32	80
11	4	3	4	2	4	3	3	2	3	4	32	80
12	4	3	4	1	3	1	3	2	2	1	24	60
13	4	2	5	1	4	2	5	3	3	2	31	77,5
14	5	1	5	1	5	1	5	1	5	1	30	75
15	5	2	4	2	4	1	5	1	3	2	29	72,5
16	5	1	4	2	4	3	4	1	5	1	30	75
17	4	1	4	3	4	1	4	1	4	1	27	67,5
18	3	2	4	1	3	1	4	2	3	2	25	62,5
19	4	1	5	1	4	1	5	1	2	2	26	65

Summe	1377,5
-------	--------

Mittelwert	72,5
Median	75
Max	82,5
Min	60
Standardab.	6,71854812

A.1.8. Results: NASA-TLX

NASA-TLX

Subject ID	Mental Demand	Physical Demand	Temporal Demand	Performance	Effort	Frustration	Total Score
1	7	1	10	4	4	1	27
2	9	2	5	4	5	2	27
3	8	4	1	12	11	2	38
4	5	2	8	5	3	6	29
5	13	14	4	6	13	8	58
6	11	5	10	13	12	15	66
7	3	1	1	5	6	1	17
8	12	2	14	7	5	1	41
9	8	3	11	8	12	1	43
10	6	1	10	5	3	1	26
11	14	5	16	12	15	12	74
12	14	2	5	5	10	11	47
13	13	3	7	7	12	7	49
14	5	2	20	1	1	1	30
15	5	1	8	13	8	6	41
16	11	1	15	7	13	2	49
17	13	2	10	8	8	3	44
18	12	4	9	12	8	11	56
19	10	1	5	5	5	2	28

Max	74
Min	17
Mittelwert	41,578947
Standardab.	14,971513

	Mental Demand	Physical Demand	Temporal Demand	Performance	Effort	Frustration	Total Score
Summe	179	56	169	139	154	93	790

A.1.9. Text for more Information

Information for the User Study: TextVision

Introduction

Welcome to our user study, conducted in collaboration between the University of Oldenburg and the German Research Center for Artificial Intelligence (DFKI). The goal of this study is to evaluate the efficiency and user-friendliness of TextVision, an innovative platform designed to optimize the document creation process. Compared to traditional word processing tools like Microsoft Word, TextVision offers an integrated environment featuring a document editor, PDF viewer, chat-based AI assistance, and specialized tools such as the "PromptDesigner." To assess our tool, tasks will first be performed in Word and then using our tool. We aim to understand how TextVision impacts your workflow.

Study Procedure

The study will take approximately 90 minutes and is divided into the following phases:

1. **Onboarding:** At the beginning, you will receive an introduction to the TextVision platform through a narrated video tutorial. Additionally, you will complete an initial questionnaire regarding your experiences with AI-assisted document creation.
2. **Task Completion:** You will perform a series of tasks designed to test the various functions of TextVision. These include utilizing AI support, the PromptDesigner, and the prompt recommendation system.
3. **Feedback and Follow-Up:** After completing the tasks, you will fill out two questionnaires to share your experiences. A semi-structured interview will also be conducted to gather more detailed feedback on specific components and features of our prototype.

During the study, your screen will be recorded. Additionally, the semi-structured interview will also be recorded. These data will help us better understand your workflow and improve the platform.

Data Protection

Protecting your data is our highest priority:

- All collected data will be anonymized and used exclusively to evaluate the TextVision prototype.
- The recordings will not include video of your face, only screen recordings and audio from the interview.
- The data will not be shared with third parties and will be permanently deleted upon completion of the project.
- Your participation is voluntary, and you may withdraw from the study at any time without providing a reason.

Compensation

For your participation, you will receive a compensation of €15 for approximately 90 minutes of your time. Payment will be made in cash immediately after the session. After the study, we will fill out a receipt together, which you will sign to confirm payment.

Consent Form

Before the study begins, you will be asked to sign a consent form at the venue. This form provides detailed information about the procedure, data protection, and your rights as a participant.

Thank you for taking the time to participate in this study. Your involvement plays a vital role in the further development of TextVision.

We look forward to your valuable insights!

A.1.10. Informed Consent Form



Informed Consent of Participation

You are invited to participate in the user study **Evaluating The Efficiency And Usability Of TextVision: A Comparative Study Of An Innovative Document Creation Interface And Traditional Word Processing Tools** initiated and conducted by . The research is supervised by Prof. Daniel Sonntag. Please note:

- Your participation is entirely voluntary and can be withdrawn at any time
- The user study will last approximately 90 minutes
- We will record personal demographics (age, gender, etc.)
- We will record videos, record audio, record the screen and/or activities on your device, and take notes during the user study.
- All records and data will be subject to standard data use policies
- Repeated participation in the study is not permitted

The alternative to participation in this study is to choose not to participate. If you have any questions or complaints about the whole informed consent process of this research study or your rights as a human research subject, please contact Prof. Daniel Sonntag (E-Mail: daniel.sonntag@dfki.de) You should carefully read the information below. Please take the time you need to read the consent form.

1. Purpose and Goal of this Research

Evaluate and compare the efficiency and usability of TextVision, an innovative document creation interface. To assess the usability and efficiency of TextVision compared to traditional word processors, gathering user feedback to identify strengths and weaknesses. Your participation will help us achieve this goal. The results of this research may be presented at scientific or professional meetings or published in scientific proceedings and journals.

2. Participation and Compensation

Your participation in this user study is completely voluntary. You will be one of approximately 10-20 people being tested for this research. You will receive 15 EUR as compensation for your participation. You may withdraw and discontinue participation at any time without penalty or losing the compensation. If you decline to participate or withdraw from the user study, no one on the campus will be told. You can deny answering questions if you feel uncomfortable in any way. The investigator may withdraw you from this research if continued participation will not meet the study goals or affect your well-being.

3. Procedure

After confirming the informed consent the procedure is as follows:

- Step 1 - Consent: Have participants read and sign a consent form detailing the study's purpose, procedures, risks, and participant rights.
- Step 2 - Onboarding: Guide participants through the TextVision interface with a video tutorial and interactive hints.
- Step 3 - Pre-task Questionnaire: Have participants complete "Basic Questionnaire 1" to assess their experience with LLMs and document editors.
- Step 4 - Task Execution: Participants perform tasks, observed while using the think-aloud method for real-time feedback.
- Step 5 - Post-task Survey & Interview: Conclude with SUS and NASA-TLX surveys and conduct a semi-structured interview using Questionnaire 2 for qualitative insights.

The complete procedure of this user study will last approximately 90 minutes.

4. Risks and Benefits

There are no risks associated with this user study. Discomforts or inconveniences will be minor and are not likely to happen. If any discomforts become a problem, you may discontinue your participation. In order to minimize any risk of infection, hygiene regulations of the DFKI apply and must be followed. Any violations of the hygiene regulations or house rules of this institution can mean immediate termination of the study. If you get injured as a direct result of participation in this research, please reach out to the principal investigator. Enrolled students are automatically insured against the consequences of accidents through statutory accident insurance and with private liability insurance in case of any damages. The confirmation of participation in this study can be obtained directly from the researchers.

5. Data Protection and Confidentiality

We are planning to publish our results from this and other sessions in scientific articles or other media. These publications will neither include your name nor cannot be associated with your identity. Any demographic information will be published anonymized and in aggregated form. Contact details (such as e-mails) can be used to track potential infection chains or to send you further details about the research. Your contact details will not be passed on to other third parties. Any data or information obtained in this user study will be treated confidentially, will be saved encrypted, and cannot be viewed by anyone outside this research project unless we have you sign a separate permission form allowing us to use them. All data you provide in this user study will be subject of the General Data Protection Regulation (GDPR) of the European Union (EU) and treated in compliance with the GDPR. Faculty and administrators from the campus will not have access to raw data or transcripts. This precaution will prevent your individual comments from having any negative repercussions. Access to the raw interview transcript and transcribed observation protocol will be limited to the authors of this research, academic colleagues, and researchers with whom he might collaborate as part of the research process. Any interview content or direct quotations from the interview, that are made available through academic publications or other academic outlets will be anonymized so that you cannot be identified. During the study, we log experimental data, record videos, record audio, record the screen and/or activities on your device, and take notes during the user study. Raw data and material will be retained securely and compliance with the GDPR, for no longer than necessary or if you contact the researchers to destroy or delete them immediately. As with any publication or online-related activity, the risk of a breach of confidentiality or anonymity is always possible. According to the GDPR, the researchers will inform the participant if a breach of confidential data was detected.

6. Identification of Investigators

If you have any questions or concerns about the research, please feel free to contact:

Prof. Daniel Sonntag
Principal Investigator
Trippstadter Str. 122
67663 Kaiserslautern, Germany
daniel.sonntag@dfki.de

7. Informed Consent and Agreement

This consent form will be retained securely and in compliance with the GDPR for no longer than necessary.

- ☐ I understand the explanation provided to me. I understand and will follow the hygiene rules of the institution. I understand that this declaration of consent is revocable at any time. I have been given a copy of this form. I have had all my questions answered to my satisfaction, and I voluntarily agree to participate in this user study.
- ☐ I agree that the researchers will record videos, record audio, record the screen and/or activities on my device, and take notes during the user study. I understand that all data will be treated confidentially and in compliance with the GDPR. I understand that the material will be anonymized and cannot be associated with my name. I understand that full anonymity cannot be guaranteed and a breach of confidentiality is always possible. From the consent of publication, I cannot derive any rights (such as any explicit acknowledgment, financial benefit, or co-authorship). I understand that the material can be published worldwide and may be the subject of a press release linked to social media or other promotional activities. Before publication, I can revoke my consent at any time. Once the material has been committed to publication it will not be possible to revoke the consent.

Printed Name of Subject

Signature of Subject

Location, Date

A.1.11. Exemplary (Deepfake) Paper used in User Study

How can humans and AI work together to detect deepfakes?

Researchers:

Matthew Groh, Ziv Epstein, Chaz Firestone, and Rosalind Picard

Associate Editors:

Christy Ascione and Fiona Firth



Abstract

Fake news is not new on the internet, and people often change images and videos for a joke. However, deepfakes aren't only meant to make you laugh. Instead, they can spread misinformation or discredit a person or a group. As more deepfakes find their way onto the internet, we need to find the best way to detect these harmful videos. We tested whether the leading AI model or humans were better

at detecting deepfakes online. We found that humans and the AI model were each good at identifying certain types of deepfakes. Maybe we could merge the abilities of both AI and humans to create the best deepfake detection model!

Introduction

Imagine you are watching a video online of your favorite celebrity and they say something very offensive. You might be shocked, or maybe even angry. But you have a feeling that something is wrong with this video – and it turns out it's a **deepfake**.

For a long time, video evidence has been the best way of indicating whether someone did or said something. Unfortunately, the rise of deepfakes means that is no longer the case. **Deepfakes are videos that show people saying or doing things that didn't really happen.** These videos are created by an **artificial intelligence (AI)** system based on **deep learning** – leading to the name deepfake. There have been a handful of deepfakes that have gone viral in the past few years. These include videos of well-known figures like Barack Obama, Donald Trump, and Mark Zuckerberg

saying things they didn't say. But can we tell the difference between real and fake videos that we see online?

Engineers have trained **machine learning** models to try to identify deepfakes on the internet. However, the best known model can only detect deepfakes with around 65% accuracy.

We wanted to know whether humans or the latest machine learning model were better at detecting a deepfake. We also wanted to see if human emotion plays a role in our ability to tell what's real and what's fake.

We then used our results to suggest how best to detect deepfakes online!

Methods

We designed a website called *Detect Fakes*, where anyone could view deepfake videos. We used this site to test how accurately ordinary people could detect a deepfake. Most

of the videos were of unknown people making unimportant statements. That was to make the experiment more equal between humans and the machine learning model.

We conducted two experiments:

→ **Experiment 1** – One real and one fake video side by side
All participants in this experiment found the site while browsing. They watched a deepfake video alongside its corresponding real video, and we then asked them to choose which one was fake. There were 56 pairs of videos. We examined the accuracy of 882 participants who watched at least 10 pairs of videos (Figure 1). The machine learning model assessed all 56 videos.

→ **Experiment 2** – One real or fake video

We had two kinds of participants in this experiment: people we recruited and people who found the site while browsing. We focused on participants who had viewed at least 10 videos. In total, there were 301 recruited participants and 1,879 non-recruited participants.

Our recruited participants started the assignment with a writing exercise. Half of them just had to write about their

day. The other half had to write about things that made them angry. This was to test how emotion impacts decision-making. They then watched the videos.

All participants were shown one video at a time. They had to share how confident they were, from 50–100%, that the video was real or a deepfake. The videos we used included four videos of Kim Jong-un and Vladimir Putin, two of which were fake. We then showed participants what the model predicted and allowed them to change their answers. This way we could see whether human decision-making was impacted by machine predictions. The machine learning model assessed 50 videos from a **dataset** of 4,000 videos.

In both experiments, we included **interventions**. These included **obstructed** faces and inverted videos – videos shown upside down. We did this to see if the way our brain naturally recognizes faces changes our ability to identify a deepfake.



Figure 1:

Here are two images used in Experiment 1. One of these images is from the original video and one is from the deepfake. We asked the participants to identify which of these two videos was the deepfake. Can you guess which image is the deepfake? Image: [PNAS](#), [CC BY-NC-ND](#)

Results

→ **Experiment 1**

The leading machine learning model correctly identified 65% of the videos in the dataset (Figure 2).

82% of participants outperformed the machine learning model.

Participants were better at detecting fakes in high-quality videos. When the video was inverted or of low quality, they were 5% less accurate.

→ **Experiment 2**

Recruited participants identified deepfakes 66% of the time.

Non-recruited participants identified deepfakes 72% of the time.

*Please see
Figure 2 on page 3*

The leading machine learning model correctly identified 80% of videos when shown 50 videos from the dataset.

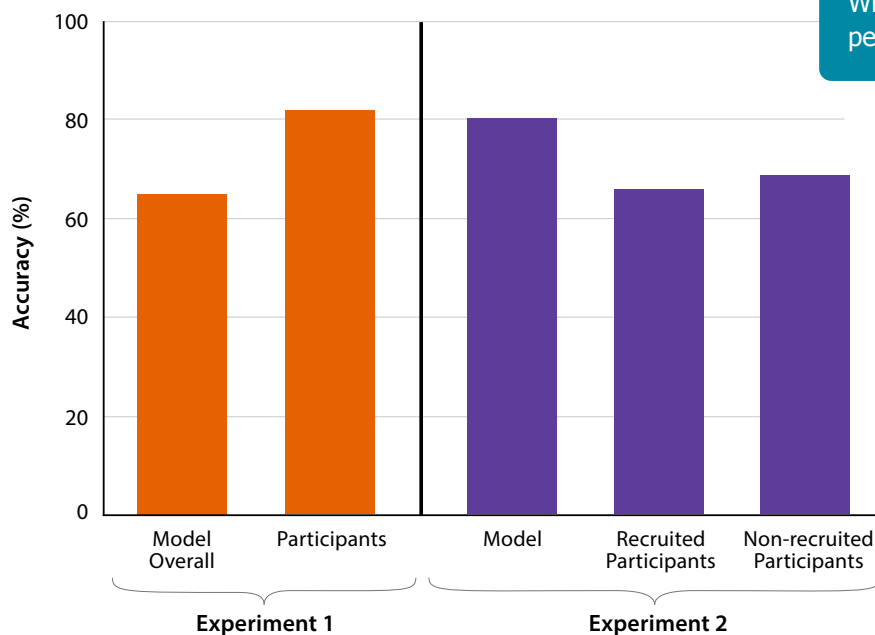
When shown the entire dataset of 4,000 videos, the model achieved 65% accuracy.

Participants who wrote about things that made them angry were quicker to judge a real video as a deepfake.

The machine learning model was better at detecting low-

quality deepfakes such as grainy, blurry, and inverted videos.

Participants were more likely to change their answer if it was different to the model's answer. This usually made them more accurate, except in the case of the political leader videos. The machine learning model was incorrect about both real videos of the political leaders. Meanwhile, 64% of participants correctly identified the videos of political leaders.



Which group of participants performed best in Experiment 2?

Figure 2.
Human performance compared to model performance

Discussion

Our results suggest that people are as good at identifying a deepfake as the leading machine learning model. Participants were better than the model when it came to the four videos of Kim Jong-un and Vladimir Putin. This could be because we can think critically about the content beyond the visual clues. The machine learning model was pretty certain that the authentic videos were deepfakes. This leads to questions about the model's ability to analyze the context of a video.

Emotion does seem to impact human decision-making by

decreasing our accuracy. Therefore, deepfakes that provoke emotion may be harder for us to detect.

In Experiment 2, participants could change their initial answer after seeing the model's response. This helped participants improve their accuracy in most cases. However, it also led them to change their correct answers. This suggests that human-machine collaboration might not lead to more accurate results.

Conclusion

The good news is people appear to be good at detecting deepfakes. The bad news is that deepfakes will likely get more difficult to detect over time. However, we can take action. Humans are better at thinking critically about the

content and context of the video. Always think before sharing viral videos, especially if they make us angry or upset! We can tell our friends and family to be careful, too.

Glossary of Key Terms

Artificial intelligence (AI) - machines and systems that can perform certain tasks. If these tasks were done by humans, they would need the thinking and learning skills we call 'intelligence'.

Dataset - a collection of data.

Deepfakes - images and videos that depict events that have not occurred.

Deep learning - pattern recognition that is based on neural networks.

Intervention - the act of purposely changing a condition within the experimental design to test something more specific.

Machine-learning model - a pattern-matching algorithm that learns from data.

Obstructed - blocked. In our research, this was something in the way of the faces, making them hard to see.

Check your understanding

- 1 How does emotion impact people's ability to accurately detect a deepfake?
- 2 What type of deepfakes were humans better able to detect than machine learning programs?
- 3 What type of deepfakes were machine learning programs better able to detect than humans?
- 4 How do you think scientists could use both AI and human detection abilities to create an accurate deepfake detection model?
- 5 Can you think of some situations where someone would create a deepfake video of a known person? In groups, discuss why someone might do this – try and think of both positive and negative situations.

REFERENCES

Matthew Groh, Ziv Epstein, Chaz Firestone, and Rosalind Picard (2021) *Deepfake detection by human crowds, machines, and machine-informed crowds*. Proceedings of the National Academy of Sciences.

<https://www.pnas.org/doi/10.1073/pnas.2110013119>

University of Virginia: What the heck is a deepfake?

<https://security.virginia.edu/deepfakes>

TIME for Kids: Fakeout

<https://www.timeforkids.com/g56/fakeout-2/>

Acknowledgment:

This article's adaptation was supported by the Akamai Foundation.



A.2. Transcripts of semi-structured Interviews

Subject 1

Transcript

00:00:01 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:10 Subject 1

I would say it was very intuitive. Especially because I already have some experience working with such tools. It was very quick, so it took me much less time than working with Word normally and I probably even got a better result in the end.

00:00:36 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:43 Subject 1

I think the highlighting and then saving to the context and then just asking the questions that fit the context and then giving me the solutions or answers that I want.

00:01:03 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:19 Subject 1

I would say that it is generally much easier and more efficient. Because normally you often have to think about how to phrase things, have I put the comma correctly and so on and so forth. And the AI helps you to see what is perhaps version a, version b, and which of the two I like better. I can also customize it myself, which makes it much easier and quicker to complete tasks.

00:02:17 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:02:30 Subject 1

Uh, yes, so it was definitely more efficient if you can get along with one screen, of course it depends on how well everything harmonizes with each other, i.e. whether I can mark my things correctly in the PDF. But that's great in itself, because then you don't have to switch back and forth, which saves time.

00:02:58 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:03:04 Subject 1

The highlighting of the PDF, but you also have this in Word or in general with any PDF reader, you have the problem that if there are any images or tables in there or any backgrounds have been designed by the PDF itself, that there are always problems. So I would generally say that this is normal and didn't bother me that much.

00:03:31 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:03:40 Subject 1

I was surprised that it did it so well and also the way I wanted it to. I would have thought that because it is connected to the API, it might do a little worse than the real AI like ChatGPT. But it did it exactly as ChatGPT would do it.

00:04:05 User Study Supervisor

Did you use the voice command feature? If so, how did it affect your document creation process?

00:04:17 Subject 1

I did not use it. But of course it is useful for accessibility.

00:05:04 User Study Supervisor

Okay then: In what ways do you think TextVision could be improved to better suit your needs?

00:05:12 Subject 1

I think you might have to make it a bit more appealing in terms of design, because it's all so "bare" and not as minimalist as I would like it to be. So it looks very loaded.

00:05:35 User Study Supervisor

But the problem is, of course, that this is due to the fact that we have the chat, PDF viewer and editor. Do you have any idea how this could be better structured visually?

00:05:45 Subject 1

Yes, maybe you could make it so that the PDF or the editor is a bit smaller at first and then when you hover over it, it gets bigger or something. That maybe you don't split it up straight away, but make the more important things bigger and maybe only make the rest bigger when you click in. For example, the editor itself. I don't think that's so important at the beginning, for example. The interaction between PDF and chat is more important. And only when you want to add things to the editor or at the end, when you've finished everything, is the editor important. So I don't think you need to see the editor so big all the time. But that's just a design thing.

00:06:33 User Study Supervisor

That makes sense. What would you think if you could collapse every single element?

00:06:44 Subject 1

Yes, that would be a good solution if you could collapse the individual elements or even the notes on the side if you don't use them anyway.

00:07:15 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:07:25 Subject 1

I don't think you need a lot of experience or a lot of effort to use it. So I think TextVision is very intuitive.

00:07:34 User Study Supervisor

So everything was relaxed and you had to make less effort to write something.

00:07:39 Subject 1

Yes, definitely. The fact that you always get a full text, which you can then see in full, means that you can see what you don't like rather than having to think about how you might like it from the start. It simply takes much longer and is much more demanding.

00:07:56 User Study Supervisor

Especially when you get used to the tool. That was your first insight.

00:08:00 Subject 1

Yes, exactly.

00:08:01 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:08:11 Subject 1

Yes, maybe if it's cheaper than ChatGPT or maybe even free with ads or something. Why not.

Subject 02

Transcript

00:00:03 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:09 Subject 02

It was definitely easier to get started than if you had a whole blank Word page in front of you, because you could just type in the prompts. And so I already knew, if I had a short summary of the text, what it was about and I didn't have to start from 0 and read through the whole text once, for example, but could already know what it was about. And then it was also very easy to use while editing or creating my text. I had the same functions as with Microsoft Word, for example. I could change the font, the font size, what was highlighted in bold and so on and so forth. The only thing I was missing was that I couldn't change the line spacing, which is what I was looking for.

00:01:10 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:01:16 Subject 02

Saving the prompts. So if I were to read things like this more often, for my bachelor thesis for example, then I can use this prompt that I have created optimally over and over again. And I don't have to think about what to ask every time, everything is saved in one place.

00:01:41 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:53 Subject 02

Yes, maybe if it's cheaper than ChatGPT or maybe even free with ads or something. Why not. And then with the suggested things that were not quite what I was looking for, but if it works like this, I have several choices of prompt recommendations.

00:02:24 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:02:34 Subject 02

Yes, it definitely made it a lot easier to work quickly, because with the Word document I always had to go back and forth between the PDF and the Word document. Here I had everything on one screen instead. And the copy icon in the chat was also very practical and very easy. And in general, TextVision definitely saved time because I didn't always have to memorize what was in the text, but was able to read everything again through the exchange with the AI. And I could then transfer all the relevant answers directly into the document and edit them.

00:03:06 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:03:13 Subject 02

Well, it probably wasn't your fault with the highlighting, but highlighting in PDFs was generally the biggest challenge.

00:03:55 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:04:00 Subject 02

As expected, he provided good answers, summarizing everything. He also did this in other words. So I caught myself working with Word in part 1 of the tasks that it was very close to the text.

00:04:19 User Study Supervisor

Did you use the voice command feature? If so, how did it affect your document creation process?

00:04:24 Subject 02

No, I did not use it.

00:04:26 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:04:36 Subject 02

As I said, the line spacing - but that's really a small story. In addition, I wasn't 100% sure what I should write in Description in the PromptDesigner. I had almost the same thing there as in the prompt field. But I thought it was good that there were so many info icons everywhere.

00:05:27 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:05:33 Subject 02

It was definitely less, as I would have had to read the whole text beforehand and I didn't have to memorize everything or paraphrase it in my head, but the AI took over for me. And as long as I don't delete them, I can access the notes at any time. You also said that you can upload multiple PDFs and work with them.

00:06:14 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:06:20 Subject 02

Yes, I would. Simply because, as I said, you have everything on one screen and don't have to switch between Word or ChatGPT or anything else. And I find it practical that you can really upload several PDFs here and don't have to switch back and forth so much between different applications or websites

Interview - Subject 03

Transcript

00:00:00 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:06 Subject 03

It's much cooler that you can customize and summarize things directly. So that's much easier with TextVision. Especially when I have to work quickly. It would be a lot easier and more efficient to work with than with Word.

00:00:31 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:35 Subject 03

I think it's really cool that you get different results thanks to the different prompts that you can also enter and that you can then adapt them to the texts that you write. I can imagine that it could be a help if a few prompts were pre-installed. So that you don't have to put together generic prompts like "Summarize" yourself first. That's also a great opportunity, especially from a computer science perspective, it's certainly great to be able to put it all together yourself, but now in my degree program, in special education, it would be really good to have something like that. if you could just look again.

00:01:51 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:57 Subject 03

Well, with PromptDesigner I thought, these are the 3 problems of my prompt, I just select them all and then all problems are solved. Then it turned out relatively quickly that this is not so good, but that you should also look carefully at which of the issues that have arisen fit the topic of the prompt. But I also believe that if I were to create a prompt in PromptDesigner from scratch, I would already have that in mind and then read it again more carefully. But that's just something normal when you try out something new, that new insights can also be gained through mistakes.

00:03:16 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:03:53 Subject 03

Yes, I think it has definitely improved efficiency as long as you have a larger screen size. I think if I had that on my small laptop, it would be too small for me too. But exactly, if you have a screen that big, then it's great.

00:04:10 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:04:14 Subject 03

Indeed, highlighting across several pages. But then I also found the solution that you can simply save this as two separate notes and then continue to use this directly without any problems.

00:05:14 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:05:22 Subject 03

This was still a bit new to me, but I found it really interesting and good.

00:05:54 User Study Supervisor

Did you use the voice command feature? If so, how did it affect your document creation process?

00:06:02 Subject 03

No, not really. But I don't really like that kind of thing anyway, it usually doesn't come across as perfectly as intended on technical devices. But it's certainly not bad for accessibility.

00:06:31 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:06:37 Subject 03

I really think that if there were more presets (like the pre-installed prompts) that I could simply select as a non-computer scientist, that would be great. And of course this PDF problem with highlighting in general. This is a general problem when working with PDFs.

00:06:59 User Study Supervisor

Exactly, we would have to see how this could be solved. How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:07:14 Subject 03

So for me it was totally relaxed, in any case, especially because I was only supposed to upload the result. I also didn't have to revise the text because I already liked it so much, but of course you can do that too. So if you had to continue working with the text of the chat, it would definitely be a great relief to have the generated text as a basis.

00:08:31 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:08:36 Subject 03

Yes, definitely, because that would simply be a huge relief and also easy if you have documents. So the PDF you used as an example was rather short, but sometimes you have, I'll just say 5 articles and all of them are over 20 pages long. Then it's a lot more relaxed if you simply have the option with TextVision to say, please write out all the things that are here on the subject of health or something similar. So that would definitely be a huge relief and TextVision even works with sources here, so it would even be comprehensible and I wouldn't have to look up all the sources myself. So it's really accurate and very easy to follow and I think that's really cool.

Interview – Subject 04

Transcript

00:00:02 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:10 Subject 04

First of all, you have a tool that helps you. You don't have that in Word. It was easier just because you don't have to keep switching tabs and also because you simply have the PDF and can type at the same time. That's already a help. And then you still have these normal things of “Hey, you made a mistake”; that is, if you have to type much at all, because you don't actually have to do that much yourself. So it's already eliminated a major source of error. It's just not really comparable because Word doesn't really do anything for you and you have to do everything yourself. You have a bit more of a feeling at the beginning: as soon as you start, half of it is already there and you don't have this blockade at all. So I generally don't have any problems with it, but you can still tell the difference that if I just type something into the chat, it's just there and then I just have to edit it. And I think it's also easier for a lot of people who have executive dysfunction, for example, so people who have ADHD or something like that, who find it more difficult to deal with, easier and yes.

00:02:04 User Study Supervisor

What aspects of TextVision's interface did you find the most helpful for document creation?

00:02:12 Subject 04

I found the layout of the interface good. From left to right, because you are used to writing from left to right. The layout, i.e. from top to bottom, as it is structured, simply makes sense. However, it would be more intuitive if the chat was at the top and the editor at the bottom. So that you can think in an arc, so to speak. So, I think it was unusual for me that the things that are highlighted are then saved to the left, but you get used to that very quickly and. The only thing I noticed is that this button only appears in the chat for the “Do you want to copy it over?” answers when you are hovering over the text with the mouse. At one point I was like, “Okay, where was the button?” I think it's always easier for me personally if all the buttons you can click are really visible in one line and are permanently there. I also found the PromptDesigner easy, because everything is simply arranged from top to bottom in the flow, so you just have to edit it in the order it's there.

00:04:17 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:04:21 Subject 04

When I was shown the PromptDesigner, I found it quite confusing and thought to myself, okay, I'm curious to see how the task will turn out when I use it. But in the end I found it quite simple; I worked on it from top to bottom, which made it easy. And then the prompt he suggested worked perfectly. All great.

00:05:15 User Study Supervisor

And the prompt recommendations, in terms of the idea; of course they weren't perfect as things stand today. But what do you think of the idea?

00:05:20 Subject 04

The prompt recommendations seem a bit out of context from my input in the chat.

00:07:30 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:07:41 Subject 04

I think it has significantly improved efficiency that it was all together like this. I think it's also just good to have a big screen. I think doing the same thing on a laptop would be really tiring. Even with myopia and everything, you wouldn't be able to read it. But yes, I really like the idea of color highlighting when you create notes. It's super practical. The fact that everything marked is simply highlighted, for example, because "Hey, you could delete this old context that's in there", and it's then marked in red, for example. These little things help to say "Hey, that's right, I don't want that context in there anymore" and that's just through the color. So I think it would be cool to have a very good overview and be able to see at a glance which resources are all available. What I would also think would be cool would be if there was a little hint in the TextVision main interface in the chat, for example, that would take you to the PromptDesigner. That way, if you type something in and the market is not so helpful and the Prompt Recommendations don't help either, you would be directed to the PromptDesigner. The PromptDesigner can also be used to learn how to learn. For example, if you ask yourself, what should prompts generally look like? Because of course you're used to human intuition when you give orders to others and you have to learn first, okay that's also a skill to enter things into the AI (prompts).

00:11:10 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:11:19 Speaker 4

Well, yes, the highlighting a bit, but I mean, highlighting text in PDF editors in general is always like this or that and doesn't work optimally.

00:11:43 User Study Supervisor

That's right. But you have actually already found the right workaround by marking the text passages individually as notes.

00:12:15 Subject 04

Maybe it would be another idea to have a small AI that first goes over the PDF and captures the text and formatting if possible and isolates it into small sections. That would be an even more optimal and simpler way. But it didn't really bother me

00:12:52 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:13:00 Subject 04

I think that's about what I expected. So that it was sometimes even a little too long or a little bit missing or a little bit done, because the entire PDF is taken as context. So the chat is more or less what I expected, just like this. But the integration with the PromptDesigner was surprising, I didn't have that on my screen.

00:14:37 User Study Supervisor

Did you use the voice command feature? If so, how did it affect your document creation process?

00:14:43 Subject 04

I didn't use it, I just don't use voice features. But it may be good for accessibility. I also personally know people who use voice features like Alexa. But I'm just not used to using it, but people who are used to it will probably use it with TextVision. I think it's just a question of type.

00:15:27 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:15:32 Subject 04

Yes, the reference to the PomptDesigner. And otherwise, what you can do with it, it does quite well. But of course it's also a bit of a question as to what exactly the task is, because I'm currently thinking about it in relation to my work (theater/writing). I was wondering a bit about this, for example, if we were to stage a novel and have the entire text of the novel as a PDF and upload it to TextVision. Then you could just say "Why did he kill the cat?". And then the answer might come out relatively quickly and hopefully a bit shorter than the novel. That would just be where I see an application in my area of life, coming from the theater. Apart from that, I don't have that much to do with summarizing PDFs or similar, but others like scientists do; for them it would be all the more super.

00:16:42 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:16:48 Subject 04

There is hardly any cognitive load. It's really easy because everything is supported and integrated into the interface and you don't even have to memorize a lot of things in the long term. It's much easier.

00:17:37 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:17:51 Subject 04

Yes, I would, if I have a suitable task for it, since, as I said, I come from the theater.

Interview – Subject 05

Transcript

00:00:02 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:22 Subject 05

Okay, I think it is really useful for everyone. And for me, especially if you are a student in university, I think it is very useful to use TextVision.

00:00:46 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:01:17 Subject 05

Yeah, I liked the PromptDesigner. It is very useful in regard to the time, since you are quicker.

00:01:31 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:02:21 Subject 05

I think my efficiency improved when working and creating documents.

00:02:46 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:02:54 Subject 05

Yeah, it definitely did. And I quickly got used to working with it and got confident.

00:03:20 User Study Supervisor

Yeah, this is normal. It is after all your first time using it. Can you describe any challenges you encountered while using TextVision?

00:03:26 Subject 05

Yeah, at the start. I was really anxious to use it. So, I asked you for help, to do the tasks. But afterwards, I was very confident to use it. It is a new thing for me after all.

00:03:57 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:04:06 Subject 05

I think I normally struggle with using AI tools and thus I'm not efficient with it. But in this tool, I was confident in using it.

00:04:50 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:04:55 Subject 05

No, it is perfect.

00:04:56 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:05:54 Subject 05

It makes you feel very slow when using it. So, with TextVision I saved time. It is very efficient for workers and students.

00:06:52 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:07:06 Subject 05

If one day I will be a researcher, let's say. Then probably, since it would be really efficient to use it.

Interview – Subject 06

Transcript

00:00:02 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:10 Subject 06

So, I mean to begin with, I cannot really fairly compare them, because we had some problems with TextVision that were mostly related to internet connection. And to be honest also I'm not used to using those kinds of tools. But I would say that TextVision was rather easy to use. Of course, it was more straightforward to complete the tasks using MS Word because you directly know what you are supposed to do in order to make a summary as an example. You are trained for that even in elementary school as compulsory education. But you have to think more and type. That's quite straightforward as a task, but of course it requires some mental demand. Now with TextVision, we weren't really able to complete all the tasks, probably due to unrelated problems with the actual tool like the internet connection. But I mean overall the idea I would say that it could be helpful. I mean generally, of what I saw, the prompts that we managed to make work, I would say that they could potentially save time in completing these kinds of tasks.

00:01:27 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:01:32 Subject 06

I mean, I would say that the overall interface is quite user-friendly. It's very easy to learn how to use it. It probably needs like an introduction or tutorial of a few minutes at the beginning. After that, it would be rather straightforward. It's quite handy, so if you compare it with the standard interface of ChatGTP in the browser, because you have everything in one window. You also have visual highlights, so I would say it is quite practical and easy to use.

00:02:12 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:02:31 Subject 06

I think in one of our repetitions we managed to get some helpful output from a prompt recommendation which gave some quite satisfactory results. So, I mean in the task with the Prompt Recommendations. But I would say that it kind of irritated me at some point, since the recommendations weren't really in the same context as my chat input. So, I'm actually very eager to see how the updated version of TextVision is, since you mentioned that it is already functional.

00:03:22 Subject 06

I don't know how the machine the model works, so it takes all the questions you give and then it tries to give you an answer or it also takes the question you give and then based on the text you have tries to improve the question.

00:04:15 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:04:22 Subject 06

I think it has the potential of greatly improving efficiency. I think it was quite easy and nice to use, especially just for users that are not really very familiar using AI or such text tools. And this was kind of like a very entry-level tool. I think it could be very good.

00:04:47 User Study Supervisor

Can you describe any challenges you encounter while using TextVision?

00:04:50 Subject 06

I mean to begin with, I had some challenges because I'm not very familiar with using LLM-based tools. So in the beginning I needed some time to get used to it. Otherwise, I had some problems with the Prompt Recommendations and the PromptDesigner, which could be improved or is already improved for the next participants.

00:06:00 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:06:09 Subject 06

Hard question, I would say when it worked the output was quite good quality.

00:06:28 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:06:37 Subject 06

Yeah, I mean, I would say that the overall interface and the graphic user interface and the functionality, in the sense of like ergonomical design of the application, looks quite good to me. I mean I could use it potentially for something, but the prompting features have to be improved. That's like one really necessary step for me to be able to use it effectively. Otherwise, it would take me more time compared to doing it manually myself in a traditional way.

00:07:36 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:08:03 Subject 06

Here comes the question, which is a question of principle. If you trust the AI tool to give you an optimal output quickly. If yes, then it would have less cognitive load. So yeah, an optimization process has to most likely be involved. But even with that the cognitive load is maybe a bit less in TextVision but some workload is still needed. Compared to ChatGPT I think the tool is a bit more user-friendly, which also helps.

00:09:16 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:09:24 Subject 06

Not in the immediate future. But in the midterm future. I could consider it, after it gets some updates.

Interview – Subject 07

Transcript

00:00:02 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:13 Subject 07

TextVision was very easy. It does a lot more, helps a lot more, actually does almost all the work for you. And it's also easy to use.

00:00:29 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:43 Subject 07

That you don't have to create the content yourself, that is, that the text can simply be summarized for you and you can then edit it directly in the program.

00:01:03 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:10 Subject 07

I think that if you need the same prompt very often, for example in different documents, then the PromptDesigner can be very helpful. Now for such a relatively simple task, I might not necessarily use it in everyday life, but if, for example, you have to summarize different papers in your studies, then you could of course use it every time. I think that would also be much quicker. I don't think I found the prompt recommendations that helpful in one of the tasks, because the PDF only described one experiment at a time; in my case, either the first or the second. But I wanted to allude to both with my chat input. But in general, I think the recommendations can be helpful so that you can perhaps formulate better what you actually want. Because I know from AI that you have to formulate very precisely what you want so that the result is reasonable.

00:02:56 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:03:09 Subject 07

Yes, I thought it was really good that you could mark and then directly select with a "note" that you should focus on that. Because otherwise you always have to select text somehow, copy it into the question and write it, let me summarize. But that makes it a lot easier with TextVision. I also found the document creation helpful, but I found that it works without it, but it's nice to have it when it's there.

00:03:45 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:03:52 Subject 07

I actually found it particularly striking that you could copy the text directly from the answer with one click, I thought that was really good. But the sources are still listed behind it somehow. I've personally always removed them because you might not always want them. So if you could maybe select how exactly you want them inserted, that would be really cool.

00:04:33 User Study Supervisor

With TextVision, you can normally upload multiple PDFs here and the PDF from which the input comes is referenced. In what ways do you think TextVision could be improved to better suit your needs?

00:05:03 Subject 07

Oh, exactly, that you could perhaps select something so that only certain passages are used. But I think that could really help with the workflow, especially with prompting. And you can also set a lot more in TextVision, like the temperature in the PromptDesigner. ChatGPT doesn't have anything like that.

00:06:08 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:06:14 Subject 07

TextVision is pretty relaxed. It makes life pretty easy. Yes, you don't really have to do anything yourself, except read it over and maybe change something. Whether that's always a smart idea is another matter. But it definitely makes life easier.

00:06:35 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:06:43 Subject 07

Yes, I would use it because it makes work easier.

Interview – Subject 08

Transcript

00:00:12 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:20 Subject 08

It was much easier, and it was also quicker overall. Although I actually found that the main advantage was not necessarily the ability to format your own document or anything like that, but rather the ability to extract text from the PDF by focusing on it with highlighting. And I don't think MS Word can keep up with that.

00:00:45 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:01:00 Subject 08

The highlighting and saving as a "note" and this direct export as a PDF.

00:01:11 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:27 Subject 08

Well, if the PromptDesigner had worked perfectly and you could use the prompts you created more often, then I think that would be positive. But now with a single task and only in one document, this advantage of reusability has not yet been used.

00:01:40 User Study Supervisor

And the prompt recommendations that appeared during the chat?

00:01:43 Subject 08

I liked those.

00:01:48 User Study Supervisor

Did you feel that having all tools integrated into one interface (document editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:01:56 Subject 08

Yes, it did.

00:02:02 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:02:09 Subject 08

Yes, that you can't directly follow up on or break down TextVision's answers into individual parts or something like that. At the moment you only have one complete answer.

00:02:44 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:02:51 Subject 08

I didn't have great expectations. I found it surprisingly good that I could use it straight away even with no basis or prior knowledge at all.

00:03:07 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:03:59 Subject 08

That you can see from the chat answers that you can somehow edit them further individually

00:05:13 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:05:19 Subject 08

I actually find it more intuitive with TextVision because you can just have this one place where you can write and also interact with the AI. And you don't have to search around in different submenus, as is often the case.

00:05:32 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:05:39 Subject 08

Yes, I think especially at university, when writing reports or homework. I think I don't have much to do with documents in my private life at the moment.

Interview – Subject 09

Transcript

00:00:01 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:09 Subject 09

So traditional in the sense of ChatGPT. I see. Faster, simpler, easier. And in comparison, much more efficient - intuitive.

00:00:55 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:01:00 Subject 09

I think the prompter is cool, the prompt position yes or the PromptDesigner, I've never really dealt with it before, like - Well, of course, I think it's trivial that you have to feed a search engine or an AI as detailed as possible to get detailed answers. But I found the prompts to see what was really missing exciting.

00:01:40 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:48 Subject 09

Oh yes, because they're not supposed to change after 3 seconds and I can read it slowly because I'm not that fast in English. Good, because in the end you only have to choose the operator yourself. What should the chat do? Should it describe something? Should it explain something, etc.

00:02:06 User Study Supervisor

Of course, the more detailed, the better.

00:02:10 Subject 09

Exactly, yes, it's definitely helpful, so you can decide for yourself whether you want to use it or not, so it's good that it's built in.

00:02:21 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:02:31 Subject 09

Yes, yes. So it's a bit like LaTeX. That's spot on, and it's also super practical, yes. In any case, very practical.

00:02:50 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:03:04 Subject 09

Yes, I would like a numbering system that goes from 1 to 5 and does not start at 1 again.

00:03:20 User Study Supervisor

Did that happen with the notes?

00:03:26 Subject 09

Yes, at the top of the editor when you enumerate. That would have to be fixed again, but otherwise it's a completely different way of working for the time being, because there's also a thought step where you have to think about how to ask the questions. And then you also have to evaluate, so to speak. OK, which of the recommendations that are suggested to me now do I actually take? But once you have adjusted to this, you work more efficiently and ultimately reach your goal faster.

00:04:24 User Study Supervisor

You've only been working with TextVision for max. 30 minutes and you must have 10 years of experience with Word.

00:04:27 Subject 09

Exactly, in Word I had a result in 30 minutes that would have taken me another hour to check. Well, the formatting wasn't quite right in the editor either, but that doesn't matter. It's all about the pure raw results and, yes, you get them faster, that's for sure.

00:05:03 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:05:15 Subject 09

Yes, yes, yes, I'm fine with it. I don't know what my requirements would be.

00:05:29 Subject 09

No, so of course, if you have several chats open now, I don't know what it's like if you want to use several documents or several of these interfaces. But I think you'll be able to do that somehow via a side tab. And then, that's, that's exactly the same, so yes, oh, you mean it's the same chat, the same chat history.

00:05:46 User Study Supervisor

Wait, as you say, with ChatGPT the links appear next to the chat. Our product has workspaces that provide such an overview in a project center.

00:05:55 Subject 09

Oh well, I could have done that here too. Okay, cool.

00:06:10 User Study Supervisor

And you can work across several documents.

00:06:18 Subject 09

That's nice.

00:06:33 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:06:41 Subject 09

Dark mode would be nice.

00:06:50 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:07:01 Subject 09

It's more tiring than in Word, but in Word I have to think about the content I want to write and in TextVision I only have to think about the operators, so as if it's much more important than what's in Word, I only care about what and I have a 1:1 representation of what I think and in TextVision I have to think about what's what and then I automatically get what's what.

00:07:58 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:08:05 Subject 09

Yes, that can help with everything where texts have to be written. So, you have to make sure that what comes out of the AI doesn't always sound too generic. That's clear, but the AI is also getting better and better. ChatGPT is also getting better and better and if this is based on ChatGPT, then it will grow with it. So you've created the interface and it then accesses the results that ChatGPT delivers, as I understand it now, quite pragmatically from there, so I would use it for term papers, theses, as I said, you have to be able to do it, you have to know how to deal with it, if you copy things 1 to 1 now, then you're rarely stupid, because then it's also somehow too blunt, but as inspiration, 100 percent. We don't have to think much in the future. We just have to know how to deal with what we get. But we don't have to think much about texts anymore. So, you still need someone to think about it. Yes, but you don't need anyone else to pull the words out of their fingers. I don't do that anymore either.

00:09:31 User Study Supervisor

That's it for the interview, thank you.

Interview – Subject 10

Transcript

00:00:03 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:16 Subject 10

Yes, it was good. So it was more relaxed than Word, I'd say.

00:00:28 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:34 Subject 10

That you could then copy the output directly. So you didn't have to mark it somehow and do something else and then you had to format it, but it was there directly with one click.

00:00:48 User Study Supervisor

Of course, you can still adjust the answers manually or something. But yes.

00:00:51 Subject 10

Yes, true.

00:00:54 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:01 Subject 10

I have to say, I've now realized that it's quicker for me if I write, give me this and that, than if I make another prompt and so on.

00:01:10 User Study Supervisor

Exactly, it's all about reusability.

00:01:18 Subject 10

If I were to work with it more and constantly now, it would of course be much more convenient if I had my ready-made prompts and only had to click on them and execute them.

00:01:37 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:01:45 Subject 10

I thought that was very cool. That's great. Because I saw that when I had to do the first task without TextVision, I kept clicking back and forth. Sure, I could have done split screen. But somehow I find it more relaxed when everything is on one screen.

00:02:09 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:02:12 Subject 10

Any complications? Yes, the one time I got the result, but that's something you still want to solve. I also found it very relaxing to use, so what I also like about the prompts is that you don't have to find the issues yourself, which you have to specify somehow, but are told directly.

00:02:40 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:03:01 Subject 10

Well, what I didn't like so much was that when you copied it, there was this 4.0 or something and all the asterisks when you copied it into the editor. That's a bit strange. I don't know if it will be solved, but yes.

00:03:19 User Study Supervisor

You mean the formatting, from copying from the chat into the editor and it's just taken over with asterisks. It's true that it's sometimes a bit annoying, because then you still have to edit it.

00:03:51 User Study Supervisor

Did you use the voice command feature? If so, how did it affect your document creation process?

00:03:55 Subject 10

No.

00:03:56 User Study Supervisor

Why not?

00:04:00 Subject 10

Didn't need it and didn't feel the need.

00:04:05 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:04:17 Subject 10

Yes, with the output, with the formatting, exactly, that it is adapted a little.

00:04:25 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:04:39 Subject 10

Ja, also mit Word ist es natürlich anstrengender, wenn ich selber schreiben muss. Und da muss ich mir das selber überlegen. Also das ist ja, da war ich natürlich ziemlich gelassen mit TextVision.

00:04:54 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:05:02 Subject 10

In any case, if the formatting is adjusted.

00:05:07 User Study Supervisor

We are pleased about that.

00:05:13 Subject 10

And I like the looks of it. Aesthetically pleasing.

00:05:16 User Study Supervisor

How do you feel about being able to refer to specific passages of text?

00:05:19 Subject 10

Very good. Yes. That I can then select them so specifically. For example, maybe the two together, that I somehow have different passages that I would like to have summarized together. Or maybe you can also write somehow, so please summarize the passage and the passage in context, I thought that was very cool.00:05:41 User Study Supervisor

00:05:42 User Study Supervisor

Thank you for this interview.

Interview – Subject 11

Transcript

00:00:01 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:07 Subject 11

Like Word?

00:00:08 User Study Supervisor

So in the first phase, where they first worked with Word and then with TextVision, what was your experience like?

00:00:14 Subject 11

So I would say that I actually find Word better, because I've gotten so used to it, to the bar at the top and the function and the whole thing.

00:00:25 Subject 11

And it's just all plain in Word, so I was in TextVision with the quotes and the asterisks...

00:00:32 User Study Supervisor

That's probably gone in the final product, because that's the markdown and the asterisks are there differently. That is, that should actually be bold.

00:00:38 Subject 11

Yes, all good, but that would just be my opinion. So exactly, I thought it was good that it was at a glance and that's just next to the PDF, but I wouldn't use it because of the editor itself. So I think Word is good, simply more user-friendly for the visuals.

00:01:03 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:01:08 Subject 11

So now also together with AI?

00:01:12 User Study Supervisor

Yes, everything you've seen.

00:01:14 Subject 11

Okay, most useful for creating documents. I think it's really cool that you can highlight something directly and then ask a question, and that you can then simply copy the answers that come up at the bottom with one click. So that was pretty cool and user-friendly. Exactly, then just at the top with the asterisks and so on. I haven't really thought about it, you say it's gone at the end, but then I think it would be a bit more visually appealing. I thought that was cool. And the function that I also found smart, but was a bit difficult to understand at first, was the one about taking notes and then integrating the note into the chat and the quick consideration. I first had to have it explained to me so that I could check it and I don't think I could have done that on my own. I think I would have puzzled over it for a long time before I copied it in. Exactly, then just at the top with the asterisks and so on. I haven't really thought about it, you say it's gone at the end, but then I think it would be a bit more visually appealing. I thought that was cool. And the function that I also found smart, but was a bit difficult to understand at first, was the one about taking notes and then integrating the note into the chat and the quick consideration. I first had to have it explained to me so that I could check it and I don't think I could have done that on my own. I think I would have puzzled over it for a long time before I copied it in.

00:02:51 User Study Supervisor

It should work and happen. The chat history is not the only context. It's also the PDF itself and more, which may be one reason why it didn't work.

00:03:13 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:03:18 Subject 11

Oh, unfortunately I found that. Unfortunately, I didn't like it at all. I had already written in a good prompt for myself that I was happy with and what he then told me always went in a different direction. So I kind of wrote, take one of the two experiments here and then it came up with experiment 1 or do you mean experiment 2, that's what it suggested.

00:03:37 User Study Supervisor

But do you mean the Prompt Recommendations?

00:03:40 Subject 11

Exactly the prompt recommendations. And then it was like, I'll summarize experiment 1. I'll summarize experiment 2. And I wanted to do both experiments together, so I was actually a bit annoyed or not really annoyed, but it took me too long, I think if I had just written it down, I would have been faster. Yes exactly, so if you already have a very specific prompt, I think, in your head, and the task is already set, ask here and ask there and if you know exactly what you want to ask, then I wouldn't use this recommendation system, but if I have no plan at all, what can I do with this text? What does it actually say? Then I could imagine that I would use it. But it was a bit frustrating for me in the task because I was like, that's already in my prompt and now just do it.

00:04:25 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:04:37 Subject 11

So I do think that it could definitely increase efficiency in the long term, especially at the moment I don't know, you're still in tune with your workflow. So I think if I had worked with my tools in this way, I might have gotten through the tasks more efficiently. But on the whole, I think it makes a lot of sense to have an overview and to work everything together. So it's not like you have a split screen where you simply have two different applications running, but one application and they all have to do with each other. I thought that was really cool.

00:05:12 User Study Supervisor

Yes, people are used to the habit, so it would presumably have been quicker with your own tools.

00:05:14 Subject 11

Yes, but in the long term, the fact that the editor somehow knows where it got it from and immediately refers to the literature or references, that I can make notes in the PDF that can then be discussed further with the AI, that's quite cool.

00:05:38 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:05:42 Subject 11

Yes, the first thing I definitely wrote down was "highlight". So that has nothing to do with text vision, but simply that there was so much highlighted in the text and I thought it was my job to highlight and I didn't find that.

00:05:54 User Study Supervisor

Yes, unfortunately this was already automatically included in the PDF, as it is a PDF of the Science Journal for Kids and Teens.

00:06:00 Subject 11

Yes, I searched for a long time until I got there, until I realized, oh, that's not the marker at all. But that has nothing to do with TextVision and the prompt input, I made a note of the fact that it took quite a long time, but maybe I missed the point a bit.

00:06:18 User Study Supervisor

With PromptDesigner?

00:06:20 Subject 11

No, with the Recommendation System. I also thought the PromptDesigner was cool, but I didn't test the issues because it clearly described what kind of prompt I needed and then I didn't look at it in detail. Should I have done that?

00:06:35 User Study Supervisor

You should read through it as there is also a potential for nonsense to come out of it or things that are perhaps superfluous.

00:06:45 Subject 11

Yes, that was a bit the case with the answers, it was all a bit too detailed for me. But I understood the idea and I think it's cool. We've already talked briefly about the fact that more, so I'll repeat it again for the interview, it's not quite clear to me why I don't just say directly in the chat, make me a decent prompt that you can work better with. But also why I create it. And if you're a statistician and you need this one, this one analysis again and again in the future, then I understand that you have one that you use again and again. I think I was a bit confused, also by the tasks. I was really working on these tasks and at that moment I didn't even remember that I had to do these "issues". Yes, it went badly, but I think it's a cool idea for the future and above all to understand what's important for an AI when it comes to prompts.

00:08:01 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:08:10 Subject 11

So if you compare it with ChatGPT, it's much more neutral and longer, more factual. I always have the feeling that ChatGPT is very personal and loose. That's not the case here, which is also okay, so you don't necessarily have to be more personal.

00:08:38 Subject 11

When I wrote a prompt (ask chat), it was always right before the text. So I think I would have found an automatic paragraph or something cool, but otherwise? I didn't have the feeling that we chatted a lot. I rather had the impression that I marked something and then I said do it and then he said, here, I did it and it wasn't like okay now do it like this, now do it like this, so it wasn't like that.

00:09:10 User Study Supervisor

In other words, virtually no togetherness.

00:09:12Subject 11

Super, yes. That's often the case with ChatGPT. You write all the time and in the end I have a result that I'm happy with.

00:09:22 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:09:30 Subject 11

That's exactly what you said. The prototypes with the asterisks and stuff and the slashes and the quotes. So simple that you don't stumble over them, visually, but that has nothing to do with the content. I think the recommendation system could perhaps guide the user a little more, so that it's like. "Hey, we have an option if you have no idea. So what if you have a text and you don't really know what to do with it? Then we also have a recommendation system thing and you can just enter something there and we'll see what we find in the text in which direction you can analyze it." Because I didn't really understand the tool at that moment. I had clear tasks and then you can maybe, I can imagine that other people might see it in the same way, that they might be really helpless and not know where to start. That a recommendation system like that would be smart, but I think you'd have to draw more attention to it or provide better guidance or something. Yes. And maybe also the temperature was not clear to me. So what I found really cool was the costs that were displayed in the PromptDesigner. It showed how expensive it was. I thought that was really cool. Yes, I think that's very smart. I think that's very cool. I think you could maybe add a little legend or something for the temperature.

00:10:59 User Study Supervisor

Behind each field there are info icons that describe the idea of the field. Even in the individual components.

00:11:03 Subject 11

Why didn't I do that? Okay, maybe the next participant is responsible this time. Spend 10 minutes on your own, click through everything, click on everything you find and then make the introduction a little shorter when you explain something and they click on it themselves first. And then you explain the most important tools. I think that way you automatically get more involved with the tool. Because I somehow have the feeling that I've been working on these tasks for so long.

00:11:30 User Study Supervisor

At the end of the second part, we said the phrase "feel free to explore the tool".

00:11:37 Subject 11

Yes, that's true. But by then an hour and a half has already passed. Well, I think, but I've seen a lot and there was just, now I haven't read the info box about the temperature, but you said how creative the point is.

00:11:58 User Study Supervisor

For scientific papers in particular, you probably won't want to do this, or at least not set it to 1, which would be the maximum, but to 0.5 or 0.6 or something like that.

00:12:06 Subject 11

Ja, ja, aber das beantwortet das die Frage ich.

00:12:10 Subject 11

Does that answer the question? I think I've gone off the rails a bit.

00:12:13 User Study Supervisor

Yes, it does. Actually references to a component. How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:12:28 Subject 11

Yes, less. But we briefly touched on that earlier. So with the first task. So when I was working with Word, I simply did a lot of thinking myself, because of course I'm kind of clear. It's also a study setting, so of course you want to do everything right, answer somehow the whole time and then you read it again, even though it's actually a super simple text.

I've already put a lot more thought into it. Of course, I also have a training effect in that I then know what it's about and now somehow also know that this paper isn't that difficult. And that's why I'm less interested in it the second time I read it, i.e. the second

task. But I often have the same thing with ChatGPT, that I usually use it as preliminary work, so that I somehow have a few ideas and super detailed and then I always don't feel like reading it through completely and then I just copy it and then somehow look at things that don't interest me that much, for example, I haven't done that now either. So I didn't read through the answers in detail, I just relied on the fact that it was correct because I had thought about it first.

Yes, I already knew what it was about and so on. So it was less of a cognitive workload, but I don't know if that makes sense.

00:14:16 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:14:23 Subject 11

Yes, I would give it a try, yes. I think I'm definitely impressed by the - Even if it's not rocket science to display all of this in one place, I think it's really cool. So you've put a lot more brainpower into the other features, I should say, but I think it's pretty good that it all works together and is linked. So, if ChatGPT, Word and PDF Reader somehow worked together, I think you would automatically be much more efficient. But it's just 3 independent things, so yes I'm impressed.

00:15:10 User Study Supervisor

Looks good, here on the 32-inch monitor.

00:15:13 Subject 11

So maybe you could make the answers a bit shorter. I always like things to be a bit more precise. It was very, very, not always, but very detailed on some points. If it were a bit shorter, it wouldn't be so much text.

But I think that will come in the future. Maybe you can also add a function like this. What do you want now, do you just want to get an overview or do you want to write a term paper, because when you write a term paper, you need pages, you need words and I think the approach you take is also very different, because for us or when I write, it always has to be as precise as possible, because we always shorten it and at university it's often the case that you always have to expand and somehow build longer sentences. And those are two different approaches. I always need it to be as short as possible and when I get texts like that, I don't read them. That would be an idea, to say up there, what do you want to write? Or do you want an overview or do you really want to produce a text that is long, short or medium? And then you choose that and then he answers you with a longer or shorter text. I think that's cool.

00:16:35 User Study Supervisor

Alright. Thank you for this interview.

Interview – Subject 12

Transcript

00:00:01 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:09 Subject 12

My experience?

00:00:10 User Study Supervisor

Yes, similar to the tool you just worked with.

00:00:13 Subject 12

My experience is limited.

00:00:16 User Study Supervisor

No, I mean the use of the two tools. How would you describe that?

00:00:20 Subject 12

Yes, that was quite good. It was a bit confusing at first, but I quickly found my way around, I think. Sometimes the task wasn't quite clear to me, but then it worked out well, I think.

00:00:35 User Study Supervisor

Yes, of course, you've probably known Word for ages and this is the first time you've seen the tool we've developed.

00:00:40 Subject 12

Yes, certainly.

00:00:40 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:52 Subject 12

Where you could create the prompts yourself (PromptDesigner). So because I simply saved myself work when you simply have one or more prefabricated things and can then simply choose what I want to do with this document. I think I liked that best.

00:01:12 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:18 User Study Supervisor

So the Prompt Recommendations were the suggestions that popped up above the chat and the PromptDesigner was the one that explicitly had its own page.

00:01:26 Subject 12

Well, it took longer to load and I found that the first prompt recommendations didn't fit so well, so I had to adjust it again. So it actually interfered more with the workflow.

00:01:47 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:01:49 Subject 12

Yes, definitely.

00:02:00 User Study Supervisor

Why?

00:02:02 Subject 12

It was clearer.

00:02:08 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:02:15 Subject 12

I think that's the case with every program. It's a little bit at first. You have to get to grips with the program first. Yes, and then the tasks were in English, so I first had to think about what I wanted to do and then I did some tasks incorrectly, so to speak.

00:02:33 User Study Supervisor

That is probably also down to us.

00:02:35 Subject 12

Yes, those were the problems. But otherwise it was good.

00:02:45 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:02:51 Subject 12

I didn't really have any expectations. It was easy to use. So it met my expectations.

00:02:59 User Study Supervisor

Maybe you think ChatGPT does it one way and here it might have been a bit different.

00:03:04 Subject 12

But it was similar. I type something in and get a result. It fully met my expectations.

00:03:20 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:03:30 Subject 12

After I have marked this, this symbol there with the options I now have with the plus and the question mark and the X or something. The symbol could somehow be a bit more concise or larger, or I don't know, something that catches the eye.

00:03:44 User Study Supervisor

Do you mean the one before the plus sign etc., i.e. the one when you go on marking something?

00:03:49 Subject 12

Exactly after I have marked it and then I can insert it into the notes on the left.

00:03:54 User Study Supervisor

Oh, yes, yes, with the green, yellow and red sign.

00:03:57 Subject 12

Yes, exactly. This window. And then, so to speak, the symbol that appears after selecting. I was also shown this in the introduction at the beginning.

00:04:07 Subject 12

I just didn't see it straight away. That would have been easier. And I wanted to mark it at the beginning, because the task said "highlight" and I really wanted to mark it with a marker and I actually use that when I read scientific texts or something, I often mark it in papers. So I think it would be kind of cool if there was a basic tool like that.

00:04:29 User Study Supervisor

Yes. That would be an idea for the future.

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:04:52 Subject 12

Yes, it was quite high. No, I don't know whether it's the program itself or simply the fact that it was a new program anyway, so it was high for me, but I'm not an expert on such things. Not extremely high, but I would say medium-high. Yes, well, I think it goes down quickly, once you've got used to the program a bit, that is, once you know the things you do with it, then it certainly goes down quickly, but at the beginning I thought oh, that's amazing. That's a lot of things.

00:05:32 User Study Supervisor

Last Question. Would you consider using TextVision for your future document creation tasks? Why or why not?

00:05:43 Subject 12

Yes, I think so. So especially with these prompts that you can save there, I already said earlier that I think that's cool. Yes, I would use that.

00:05:52 User Study Supervisor

We've reached the end. Thank you very much for the interview.

Interview - Subject 13

Transcript

00:00:02 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:17 Subject 13

I think it's easier to summarize things in a short time because you don't have to type them all out by hand. And once you get used to it, it's also quicker than typing everything manually.

00:00:40 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:47 Subject 13

Clearly the summary of selected paragraphs. And that you can give very precise instructions as to where things should be summarized.

00:01:05 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:13 Subject 13

Because it was the first time, I would say now rather handicapped because I am not familiar with the procedure. I could imagine that it can generally be helpful if you work in an area where many similar sources appear. But it's also not necessarily helpful if you're dealing with different types of texts or documents, because then the same ready-made prompts can't be used in many different ways.

00:01:48 User Study Supervisor

Of course, you can also make the prompts you create generic. So you can make generic prompts.

00:02:00 Subject 13

Yes.

00:02:03 User Study Supervisor

Now let's move away from the experiment as in the task. So it always depends on what you get, depending on what you write in. So for reusability, it's good to create prompts that are more general but still specific.

00:02:18 Subject 13

Well, then perhaps one more time. To be more precise, I think the prompts can generally be used well for different types of sources, texts and documents. But if you want to give a bit more precise instructions, then I don't think they are necessarily useful across disciplines. Yes, that was my impression.

00:02:54 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:03:04 Subject 13

Yes, definitely. That's what I would have said, because it's relatively easy to look around on the main page. So which function did I just want to use and in which corner do I have it now? Everything is also relatively well structured. There's actually no need to gesticulate, because you can't see it when you're transcribing anyway. But the fact that certain functions are inserted at the top right or side left, for example, means that everything is nicely organized so that it's easy to find, especially once you've got used to what's where.

00:03:43 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:03:51 Subject 13

Today it was actually my own attention span, meaning that I had to read the tasks over and over again and therefore worked on a very small scale. So first I marked them, then I went back over and looked to see what the task was and. I worked it out step by step by searching through the interface myself, just looking through it, that's the nice thing about it, that it's so well structured, or I simply asked you for help.

00:04:32 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:04:41 Subject 13

I didn't really have any expectations because I didn't know what to expect. And I was surprised at how quickly the results came out and also how much can be generated in a short space of time, much faster than you could type manually or even than you would probably need to process everything yourself in the first place.

00:05:15 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:05:26 Subject 13

And when the results of the chat are transferred to the editor, that then the. Yes, things like bold print or bullet points or probably other things that are in the programming that the chat then converts, but the editor doesn't yet. That this is still being adapted, that you don't have to change the formatting yourself afterwards, but that you can simply edit as you see fit. With highlighting, for example, or something like that. But apart from that, I found it very clear. The transfer also works almost 1 to 1 from the structure. It was also practical in general, the way things were output. They were very, very easy to use as point 1 or sentence 1. And then came the next paragraph or the next point and sometimes with a small gap in between, depending on the situation. In that respect, I would say. And there's not much to improve. And the info boxes for the individual functions, for example when you create the prompt. I haven't even read through them, so I don't know whether they make sense because you explained everything beforehand. But I think if you download it yourself now. Without someone explaining how to do it beforehand, it would be helpful to have a mini content specification, i.e. what belongs in there or what the field is useful for.

00:07:28 User Study Supervisor

We explained it verbally ourselves, but I am. Yes, you could also provide further assistance, such as a user journey where you are always on the page at the beginning, click through a bit, so that you are led in certain directions, in chronological order.

00:07:43 Subject 13

In a kind of tutorial. Either that, or you make a small extra point. Also in the side menu, with help if you like.

00:08:03 User Study Supervisor

That already exists with the info icons and the FAQ page. And that's simply the question of how a FAQ always helps everyone to use it correctly. Of course, if you only see it in text form and not in visual form, but most people learn by doing anyway.

00:08:18 Subject 13

Yes, that was the case and when you create prompts, the windows or this section still has to be filled in, which the prompt should actually fulfill. And if I just want to create something quickly for a few texts that I'm currently editing, then I don't necessarily have to write down again what it's actually intended for (Goal field), for example. It's a bit like that. I want to write a text and what do I want to achieve with it? So I look for a goal and when you think like that, you think, OK, what do I want to achieve with the prompt? And what's the best way to get there? But if you assume that you're more familiar with computer technology, or perhaps you're already familiar with the system and are working with an AI-supported tool, then you no longer think like that. So the handling of the program is simply different.

00:10:05 User Study Supervisor

Right. Now that you know, you'd probably be fine with the prompt and the field, right?

00:10:10 Subject 13

I think so too, yes.

00:10:19 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:10:30 Subject 13

Because it was new stuff that I'd never heard of or tried before, I had to concentrate a bit to remember it all. But it was quite good that you did it in the order that you explained the basic functions of Textvision to me at the beginning of the user study and then I did the Word tasks because I had such a small time window in between to be able to process it all.

00:11:01 User Study Supervisor

We also asked the first participants whether this makes sense in this order. And we received positive feedback.

00:11:08 Subject 13

Because the other way around, if I had worked with Word first, I would have just done everything the way I'm used to because I already know it and then I would have gotten a lot of new information that I would have had to understand and process first and that would probably have slowed me down a bit more to concentrate on it.

00:11:34 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:11:42 Subject 13

I can answer that with a resounding yes, because I found it very, very practical how easily and quickly you can have longer texts summarized and even summarize them yourself. But you can also choose whether you want to focus on certain points or determine the exact scope, for example, how deeply you want to summarize or list them. That is a huge relief. And then you can concentrate your own capacities on more demanding questions that the AI perhaps cannot or should not take on. And it also takes much less time than having to read and then prepare everything yourself and perhaps print it out beforehand and mark and write it out yourself.

00:12:34 User Study Supervisor

It is always good to create a basis.

00:12:36 Subject 13

Yes.

00:12:38 Subject 13

So let's get back to the point. If it were available as a finished product, I would definitely use it privately, even free of charge, like Word or Open AI or something like that.

00:12:55 User Study Supervisor

That was the interview. Thank you very much for your participation!

00:12:57 Subject 13

Well, thank you for the interview.

Interview – Subject 14

Transkript

00:00:03 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:16 Subject 14

What I believe is this. To search for things or anything. It's very easy in TextVision compared to Microsoft Word. We have to go to the Google window, browser window and type something in and get the results. Then we have to copy and paste them. But when we use TextVision, we do not have to do so many tasks. We just have to go to the TextVision screen. And we can just type in whatever we want to search for and get the results. We just have to click on a button, text to editor, then the text is in the editor and we can edit it to suit our needs.

00:00:57 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:01:06 Subject 14

The PromptDesigner. The first thing. Because we have different parts and during the creation of a document we have different contexts. Like introduction, decision results, conclusion methods, this, this, this and this. Here when you write an abstract because in my belief we are writing scientific papers in my course and we have to make. Our abstract must be very nice and creative. Otherwise the reader won't want to access to read the whole document. In my belief. If your scientific paper and like the abstract is good. Only then the reader will continue to read your paper. Otherwise they will just give you the grades on your abstract and when we use TextVision we can make a very nice abstract because we have multiple screens at the same time.

And the good thing that I like the most is the editor. We have the editor on the same place, we can edit as we read. So it's good. The editor part was the best thing I found.

00:02:40 User Study Supervisor

So in your opinion the PromptDesigner was the best feature, right?

00:02:41 Subject 14

Yeah, the PromptDesigner was the best part.

00:02:43 User Study Supervisor

Was it just the PromptDesigner? Or the prompt concept in general.

00:02:51 Subject 14

No, as we have this kind of thing in ChatGPT is on the left. OK? But when we make a prompt, we only make prompts for the things that we need. But while we are using chatgpt, if we make an unnecessary mail or something, it will also appear there. We have those things there and that is very annoying in chat. Which we don't have here. And we have to scroll down to find the good stuff, which makes it unclear. But when we make prompts, we just have to click on the ones we have previously made, like when we made the experimental setup prompt, we just have to think about experimental setup and have it and save it. It's good like that.

00:03:31 User Study Supervisor

I mean you mentioned the left side thing in ChatGPT which is the overview of your interactions there but with our product you can create workspaces. I mean you didn't see that because we already created it for you, but you can like share workspaces and work with other people and stuff like that. The point is that you can create workspaces for different scenarios.

00:03:47 Subject 14

Oh, so we can also share these things with others.

00:03:55 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:04:08 Subject 14

So as I mentioned earlier. We have unnecessary things in other artificial intelligence sites/applications, but in this we can easily access the prompts because we name them and we remember them. That's the most important thing, and when you make a prompt, I think it's very important to be clear about the name so that we don't have any problems in the future. And make simple names that you can remember. Don't do it like underscore, underscore, underscore because that's very annoying.

00:04:47 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:04:48 Subject 14

Yes. The only thing I don't like about TextVision is that it's slow, sort of, but I think it will be very fast when we have it on our home desktop, not on yours. Yeah. And as you mentioned, we have this problem just because of this room.

00:05:16 User Study Supervisor

Yes, that is true. It works perfectly well at home on our own computers.

00:05:25 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:05:31 Subject 14

Challenges... There are no challenges.

00:05:33 User Study Supervisor

Like problems or issues you had.

00:05:35 Subject 14

I had none, no. You introduced me to the application and maybe if you make a 5 minute video like you described it to me. I think there will be a video on YouTube or whatever where we just watch this video for 5 minutes and we know how to use this and its done. And if we play with it for like 30 minutes. I think we can enjoy that for the rest of our lives.

00:06:03 User Study Supervisor

Yes, and that's the point. Like Microsoft Word, you have probably been using it for the last 10 years, whereas TextVision is just a first look at a new application.

00:06:08 Subject 14

Yes, this is the first look. But with this product, the future will be bright.

00:06:18 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:06:22 Subject 14

What?

00:06:22 User Study Supervisor

The chat features, how do they compare to your expectations? For example, do you think ChatGPT does it in a different way that you didn't see in TextVision? Was there anything different?

00:06:35 Subject 14

I don't have to mention anything else because I'm in the biology department. So when we read a research paper, there are lots of little details. But they are very important, like we have a light response curve or something. The shape of the curve, like its slope, its hypothetical, whatever it is, is very important. And when we read this on the Google and the Microsoft Word separately, we can easily forget the names because we have many tabs and details, not a single window like you have created. So we make a lot of typos. So many typos that even we don't know we've made, and we read them over and over again. But in this we have the same screen. We have the split screen in our desktop now, but I don't think it's good because you still make typos. This is a better version, yes.

00:07:40 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:07:46 Subject 14

I'm not a technician or anything like that, so I can't say anything regarding that. But for me it's a great tool. I am waiting for the tool. When it will be on my browser and I will use it. The technical things, I don't know that much about data science and artificial intelligence. I know how to use it for my own benefit and that is all.

00:08:14 User Study Supervisor

Perhaps some features?

00:08:16 Subject 14

No, I think they're enough and better than ChatGPT, so I'm good to go.

00:08:28 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:08:38 Subject 14

It's OK. I didn't have to do anything. Easy. Even if we'd had a party, we can get our work done with this tool. What more do you want?

00:08:57 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:09:01 Subject 14

Yes, definitely. I need this tool, yes.

00:09:08 User Study Supervisor

We'll work hard and release it for you.

00:09:16 Subject 14

I think it's a big problem for us (in the biological field).

00:09:18 User Study Supervisor

Thank you very much for the interview.

Subject 15

Transcript

00:00:01 User Study Supervisor

Okay, do you agree to the interview being recorded? And should I translate the questions into German?

00:00:06 Subject 15

Yes, I agree. You don't need to translate that.

00:00:09 User Study Supervisor

Okay, perfect. How would you compare your experience using TextVision to traditional word processing tools like MS Word?

00:00:19 Subject 15

I found it to be more controlled and somehow more supportive, the prompts gave you more of a clear guideline so that you could simply pull the things you wanted out of the text - which I actually found quite helpful. And you don't have to put in this mental process of how to formulate things. Instead, you just look at what the answer was and see whether you can use it or not. I think that's how I would describe it, it was more relaxed.

00:01:02 User Study Supervisor

What aspects of TextVision's interface did you find the most helpful for document creation?

00:01:08 Subject 15

I actually thought this PromptDesigner was really cool, that you can always go back to the prompts if you have saved one there. I think that would be really cool, especially in the long-term process, if you could always go back to it and not have to think about it again and again. I think that would be really interesting for a lot of papers. And that by focusing on the text you could always refer to specific parts of the text. That way you could also reference more specific things and get answers.

00:01:53 User Study Supervisor

How did the PromptDesigner and the prompt recommendation features impact your workflow?

00:02:07 Subject 15

I think I found the latter a bit confusing in parts. That was a bit of a problem for me, because I had to think about whether this is what I actually want or whether I wanted to go in a different direction. But I think the idea behind it is good and especially the fact that you can somehow have your own prompt evaluated again and then at the same time get suggestions for improvement in the PromptDesigner. I found that really helpful because it meant you could refine it a bit yourself and I imagine it would be very helpful for people who have no idea or little experience of how to generate good prompts because it is quite intuitive.

00:03:10 User Study Supervisor

Did you feel that having all tools integrated into one interface (DocumentEditor, PDF viewer, Chat) improve your efficiency)

00:03:18 Subject 15

Yes, definitely. I don't see any disadvantage because no. Well, if you want to complain at a very high level. Then it might just be that the text field that you end up with only takes up a quarter of the screen.

00:03:43 User Study Supervisor

Yes, we are aware of that, but the problem is how to solve it. Can you describe any challenges you encountered while using TextVision?

00:04:07 Subject 15

I think the general challenge is using it for the first time and having to find your way around a bit. Exactly, but I think that's just part of it.

00:04:34 User Study Supervisor

How did the chat's capabilities compare to your expectations?

00:04:55 Subject 15

Yes, to be honest I haven't seen much of a difference. If anything, it's because of the formatting, which is still a little different.

00:05:40 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:06:00 Subject 15

I think that's difficult if you're not really that deep into the subject matter and haven't looked at it for that long. But the question might be how well I could use it if I had to work with different documents in the program over several weeks or months. Whether I could then access my files and chat histories somehow, for example, something like that.

00:06:30 User Study Supervisor

Yes, that's possible. We have something like workspaces. I created one for you straight away. So that was your realm, so to speak, where you had just worked on the PDF. And you could then go back there to create a new workspace. Exactly, and you could also upload new PDFs there, although you can of course work with several at the same time in the same workspace if you want to. And in each workspace you can then work on it across PDFs, so to speak.

00:07:27 Subject 15

Oh, that's very cool. Then maybe more precise references, because that's what you can then be more precise about.

00:07:35 User Study Supervisor

We actually have that too, so you always know which PDF the texts came from, for example. How did you feel the cognitive load of using TextVision compared to traditional word processing tools like MS Word?

00:08:25 Subject 15

No, it was significantly lower in TextVision because of the AI, of course.

00:08:49 User Study Supervisor

Okay, would you consider using tax vision for your future document creation tasks?

00:08:55 Subject 15

Yes, definitely yes. But I would use that.

Subject 16

Transcript

00:00:02 User Study Supervisor

Okay, do you consent with the audio recording?

00:00:07 Subject 16

Yes, I do consent.

00:00:07 User Study Supervisor

Okay, perfect. How would you compare your experience using text vision to MS Word?

00:00:21 Subject 16

The first part (MS Word) was that I wanted time for that, but it was an easy task actually. And the second part (TextVision) was easy since I wasn't doing the job, the AI was doing the job. It was so easy. I found it interesting, and people need more tools like this in the future. But TextVision could need some more support tools like for e. g. image analyzation, that would make it even more efficient. Also like some auto-correct function in the chat when I write some grammatical errors. I usually type with a lot of typos, so I need something like this.

00:01:54 User Study Supervisor

And what aspects of TextVision's interface did you find the most helpful for document creation?

00:02:01 Subject 16

Yeah, I really liked the PromptDesigner. That was really helpful. But I didn't have too much time to explore that since this was a time-capped study and my first time using the tool. I found it very interesting and there are quite a lot of ways to use it.

00:02:06 User Study Supervisor

And there is of course also the aspect of reusability. In this study you were only able to use it regarding its prompt optimization process, but the reusability is another big point of it.

00:02:07 Subject 16

Yeah, especially when you actually use it on multiple different PDFs with a similar use case, you would be able to also use the same prompt.

00:02:58 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:03:04 Subject 16

Oh yeah, I found it a bit complicated. I mean the prompt recommendations since they weren't completely aligned with what I was typing or intending to type. That probably needs to be optimized later on. But I believe it will be very helpful if it works well and is precise.

00:03:43 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF Viewer, Chat) improved your efficiency?

00:03:50 Subject 16

Yeah, absolutely. They have improved my efficiency. At the first part of the tasks with MS Word I had 30 minutes and couldn't completely finish my tasks. I could only complete 2 tasks of it while in regard to TextVision I could complete everything and even had enough time left to explore the other features.

00:04:21 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:04:28 Subject 16

The challenges were that you of course explained the tool to me, but when I was using it alone, I couldn't quite figure it out and had to call you to me again. But that's not an issue, people will get used to it, once they use it one or two times

00:05:21 User Study Supervisor

How do the chat's capabilities compare to your expectations?

00:05:26 Subject 16

Yeah, the chat was in a way that I didn't find any issues with that. It was working well.

00:05:57 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:06:03 Subject 16

Yeah, like I said previously, integrating more tools like image analysis. ChatGPT as an example still does a lot of mistakes when I use it with images and I'm not always trusting it. But I recommend adding a good version of image analysis to TextVision later on. Also prompt recommendations need to be improved and maybe the interface design could also be a improved. Like maybe a bit more aesthetic.

00:07:16 User Study Supervisor

How do you feel the cognitive loads of using text vision compared to MS Word?

00:07:25 Subject 16

Yeah, that was good (...).

(Digressed from the question; talked about image analysis as a feature)

00:09:20 User Study Supervisor

And would you consider texting for your future document creation task?

00:09:29 Subject 16

Yeah, probably. At the moment I think the output is quite good. Obviously, I will use when it's the end product. So, in the end product when it is probably more reliable, I will surely use it.

00:10:18 User Study Supervisor

Okay, perfect. We're done.

Interview – Subject 17

Transcript

00:00:02 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:13 Subject 17

Very good, so everything worked well, it simply took a lot of work off my hands, this program, this TextVision. Yes, I can't complain about that.

00:00:28 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:38 Subject 17

I don't know if that's part of it, but the one where it was pulled up directly (add to editor). Yes, I thought that was very good. And also that you could theoretically insert the notes directly there (mark as context). And I also thought it was really smart to be able to ask questions directly.

00:01:05 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:10 Subject 17

What I found really interesting was the issues. So what could be improved. Then it was also somehow directly improved, when you clicked on it and so on, it was somehow completely easy to use and helped a lot. Well, I think it was fun, in a way. Yes, it was good.

00:01:27 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:01:40 Subject 17

Yes, of course. So I think it's good to have everything at a glance. So it was kind of well done, so clear/visible, yes.

00:01:52 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:02:02 Subject 17

I think if you know less about the whole thing than I do, then you might need a bit longer. I think, for example, my parents or something like that, they would need a bit longer, but it was actually the case that they would manage it eventually. So it was well done and you eventually came to terms with it. Few challenges.

00:02:25 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:02:33 Subject 17

Yes, it was good.

00:02:47 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:02:55 Subject 17

I can't think of much off the top of my head. Yes, I wouldn't know where to start or what I should say specifically, what could be done better. So nothing.

00:03:17 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:03:26 Subject 17

Yes, a lot less, so that took a lot of the thinking out of it, which is much more relaxed. And I was also much quicker, basically. Yes, that's really good.

00:03:43 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:03:46 Subject 17

Sure yes, full so I would definitely use it. And because it simply reduces the amount of work and why not? that's always good.

00:04:02 User Study Supervisor

Yes, that's right. That's it for the interview. Thank you very much.

Interview - Subject 18

Transcript

00:00:05 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:12 Subject 18

So of course it's a relief because you don't have to type yourself, or except for editing. And of course the first task involved reading the text, but apart from that it was of course faster. But if that had been something for the university, then I would of course revise it considerably if I wanted to submit it in that form. Otherwise uncomplicated.

00:00:39 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:50 Subject 18

I liked the fact that you could mark things. And that you could display the questions directly or with the context (add a note, mark as context). That was really cool. So, when I finally managed to do that, yes.

00:01:09 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:18 Subject 18

Yeah, it took me a bit to figure it out, to be honest, but.... I don't know how often I'll use it, but it was actually pretty cool. Yes. But I think that was the one feature I was most unsure about.

00:01:47 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:01:56 Subject 18

Yes, yes, I normally go back and forth between the windows, but it was definitely good to have everything in view at once.

00:02:09 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

00:02:18 Subject 18

Oh, I don't know. That's not a challenge. Now when I say the product quotes itself or the thing, that was actually good. Or rather, it looked a bit awkward because it was just the name of the document, but challenge? I would say on the whole maybe, I'll have to think about it. Because I think when you're really into it, it's easy. And then it really makes sense. Now it was a bit like that, the first time, but it is, yes, but otherwise? nothing at first.

00:02:46 User Study Supervisor

Yes, that's quite normal.

00:03:00 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:03:08 Subject 18

I mean AI is already pretty much in so through ChatGPT et cetera. Now not much different, but I liked so the function (button - add to editor) of that you could directly take over the one, for example yes.

00:03:27 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:03:36 Subject 18

Good question. So it's often the case that these typical AI sentences start with "oh yes, of course I can do this here and so" and if you then take it over directly, you have to take it out at the top, which of course doesn't take long, but it's still an aspect. For example, in the last task you had to choose between two options and then it takes time again. Even if it's about 5 seconds here, but you just had to select what you wanted and add it manually and you couldn't just say OK send my selection to the editor or that actually works, but then you would have to delete a lot but yes.

00:04:21 User Study Supervisor

So I think that's what's coming back with new features. So with the development at the moment, yes. How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:04:45 Subject 18

I would say that with the Word task, the input was of course a bit newer. But otherwise it was much easier. And to be honest, I didn't work on it that much in the end. Or yes, if you really wanted to use it for any kind of submission, etc. in a university context, then you would definitely have to work on it because it's AI language. But yes, just having it there for input and then editing it - it's actually relatively easy in comparison, yes.

00:05:24 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:05:33 Subject 18

Yes, I really didn't think it was bad. So it was definitely helpful, also as I said, that you could mark things and ask the question directly et cetera. And it's even better if you have to familiarize yourself with it somehow.

00:05:50 User Study Supervisor

Perfect. Thank you very much for the interview.

Interview – Subject 19

Transcript

00:00:04 User Study Supervisor

How would you compare your experience using TextVision to traditional word processing tools like Microsoft Word?

00:00:12 Subject 19

Yes, of course it was much easier once you understood where to find everything (orientation). So for me, MS Word was easy to navigate at the beginning, but otherwise it was still good and easier.

00:00:23 User Study Supervisor

Of course, this is also due to experience, as you have probably been using MS Word for about 10 years.

00:00:27 Subject 19

Yes.

00:00:38 User Study Supervisor

What aspects of TextVision's interface did you find most helpful for document creation?

00:00:44 Subject 19

In the interface? Well, the fact that everything was on one page was good, of course, because you could see everything. The fact that you could see everything on one screen

00:00:57 Subject 19

Yes, it was actually quite easy to navigate. So I didn't have to search much to get my tasks done.

00:01:05 User Study Supervisor

How did the PromptDesigner and prompt recommendation features impact your workflow?

00:01:12 Subject 19

So when I understood it, it was good. I started looking for where it was (the PromptDesigner), but I think it's a really good idea. I tend to just enter 1-3 keywords and then hope that it spits out the right thing and that was really good here. And the displayed suggestions (from PromptDesigner and Prompt Recommendations) were also good.

00:01:36 User Study Supervisor

Did you feel that having all tools integrated into one interface (Document Editor, PDF viewer, AI-Assistant) improved your efficiency? Why or why not?

00:01:45 Subject 19

Yes, yes, I had copied the tasks from the task sheet into the editor so that I could still see them. So I had everything on the screen. That was perfect.

00:02:01 User Study Supervisor

Can you describe any challenges you encountered while using TextVision?

Perhaps there were no challenges?

00:02:11 Subject 19

I could not mark the texts in the PDF file (component) correctly. That really bothered me.

00:02:13 User Study Supervisor

Oh, yes. How did you solve it then?

00:02:18 Subject 19

At first, if it went over several pages, I always marked the individual passages, e.g. in the uploaded file I first marked experiment 1, then the first part of the second experiment and then the continuing part, which continued on the next page, individually marked, so that I had 3 parts (notes).

00:02:22 User Study Supervisor

I see, you added a note on one page and another one on the next.

00:02:36 Subject 19

And yes, that was a bit annoying, and when I copied something from the chat into the editor, the text was sometimes black.

00:02:45 User Study Supervisor

Black? In what way?

00:02:47 Subject 19

It was completely black. So I had to change the background color so that you can read it again and it is no longer blacked out.

00:02:52 Subject 19

So if I just copied it directly from the chat, i.e. manually and did not use this function “add to editor”, if I copied it manually, then the text in the editor was completely blacked out.

00:03:04 User Study Supervisor

Was the formatting and everything black?

00:03:05 Subject 19

Yes, yes, but I could change that relatively easily. But yes.

00:03:09 User Study Supervisor

We haven't had that before. That is interesting. We need to look at that again.

00:03:30 User Study Supervisor

How did the AI-Assistant's capabilities compare to your expectations?

00:03:37 Subject 19

It was easy to recognize that it was an AI. The structure of the answers was very clear and followed a certain pattern, with an introductory sentence and a final conclusion. This division was noticeable, but overall I thought the information was well presented.

00:04:01 User Study Supervisor

In what ways do you think TextVision could be improved to better suit your needs?

00:04:07 Subject 19

What I just said, yes, with the difficulties with the black highlighted text when manually copying from the chat. Other than that, I have no suggestions for improvement. I found the PromptDesigner to be the best.

00:04:24 User Study Supervisor

How do you feel the cognitive load of using TextVision compared to traditional word processing tools?

00:04:33 Subject 19

So I think the fact that you did the Word task beforehand meant less, because you already knew the text, but otherwise of course less (cognitive load), because you didn't have to write the summary yourself.

00:04:52 User Study Supervisor

Would you consider using TextVision for your future document creation tasks? Why or why not?

00:05:00 Subject 19

Yes, as I said, I think this PromptDesigner is very good. If I have many different search documents, I can easily enter the important points and compare them with each other. That way I can apply the information to all the documents. And yes, I think that's what makes it special for me. Most of the other functions can also be found in ChatGPT.

00:05:29 User Study Supervisor

Thank you very much for the interview. The recording is now finished.

A.3. Troubleshooting and Logging

A.4. Fine-tuning script

The following script is applicable for the model Mistral-7b-Instruct-v2, but can easily be used for the Llama-3-8b-16k model, by merely changing its huggingface path.

```

1 from datasets import load_dataset
2 from huggingface_hub import login
3 import evaluate
4 import torch
5 from torch import device
6 from transformers import TrainingArguments, Trainer,
   DataCollatorForLanguageModeling, AutoTokenizer, \
7     AutoModelForCausalLM, TrainerCallback
8 from peft import LoraConfig, get_peft_model, TaskType,
   prepare_model_for_kbit_training
9 from torch.distributed import init_process_group
10 import os
11 import sys
12 import random
13 import datetime
14 import socket
15 from torch.nn.parallel import DistributedDataParallel as DDP
16
17 sys.stdout = open('finetuning_inst_log.txt', 'w')
18 sys.stderr = open('finetuning_inst_log.txt', 'w')
19
20 login(token="YOURTOKEN")
21
22 torch.cuda.empty_cache()
23 os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "max_split_size_mb
   :512"
24
25
26 def setup_distributed():
27     world_size = int(os.environ.get("WORLD_SIZE", 1)) #
   Dynamically detects world size
28     local_rank = int(os.environ.get("LOCAL_RANK", 0)) #
   Local rank for GPU assignment
29
30     if world_size > 1:
31         print(f"Initializing process rank {local_rank} (world
   size: {world_size})")
32         print(f"Hostname: {socket.gethostname()}")
33         print(f"MASTER_ADDR: {os.environ.get('MASTER_ADDR', '

```

```

34         not set'})}")
35     print(f"MASTER_PORT: {os.environ.get('MASTER_PORT', '
36         not set'})}")
37
38     init_process_group(
39         backend="nccl",
40         init_method="env://"
41     )
42     device(local_rank)
43     print(f"Process rank {local_rank} initialized
44         successfully")
45 else:
46     print("Single GPU detected. Skipping DDP
47         initialization.")
48
49     return local_rank, world_size
50
51 class GradientCheckpointingTrainer(Trainer):
52     def __init__(self, *args, **kwargs):
53         super().__init__(*args, **kwargs)
54
55     def _prepare_model_for_training(self, model):
56         if isinstance(model, DDP):
57             print("Model is wrapped in DDP. Skipping gradient
58                 checkpointing enable.")
59             return model
60
61         if hasattr(model, "gradient_checkpointing_enable"):
62             print("Enabling gradient checkpointing inside
63                 Trainer before DDP wrapping...")
64             model.gradient_checkpointing_enable()
65
66         return model
67
68     def train(self, *args, **kwargs):
69         self.model = self._prepare_model_for_training(self.
70             model)
71         return super().train(*args, **kwargs)
72
73 class LossLoggerCallback(TrainerCallback):
74     def on_epoch_end(self, args, state, control, **kwargs):

```

```

70         if state.log_history:
71             last_log = state.log_history[-1]
72             if "loss" in last_log:
73                 print(f"Epoch {state.epoch}: Loss = {last_log
74                     ['loss']:.4f}")
75             else:
76                 print(f"Epoch {state.epoch}: Loss not found
77                     in logs.")
78         else:
79             print(f"Epoch {state.epoch}: No log history
80                 available.")
81
82 def load_model_and_tokenizer(local_rank, world_size):
83     print("Loading tokenizer...")
84     tokenizer = AutoTokenizer.from_pretrained("mistralai/
85         Mistral-7B-Instruct-v0.2", use_fast=True)
86     if tokenizer.pad_token is None:
87         tokenizer.pad_token = tokenizer.eos_token
88
89     print("Loading base model...")
90     base_model = AutoModelForCausalLM.from_pretrained(
91         "mistralai/Mistral-7B-Instruct-v0.2",
92         torch_dtype=torch.bfloat16,
93         low_cpu_mem_usage=True
94     )
95
96     if hasattr(base_model, "gradient_checkpointing_enable"):
97         print("Enabling gradient checkpointing on base model
98             before DDP wrapping...")
99         base_model.gradient_checkpointing_enable()
100
101     print("Preparing model for low-rank adaptation...")
102     base_model = prepare_model_for_kbit_training(base_model)
103
104     print("Setting up LoRA...")
105     lora_config = LoraConfig(
106         task_type=TaskType.CAUSAL_LM,
107         r=16,
108         lora_alpha=32,
109         target_modules=["q_proj", "v_proj"],
110         lora_dropout=0.05,
111         bias="none"

```

```

108     )
109     model = get_peft_model(base_model, lora_config)
110
111     model.train()
112
113     print(f"Moving model to GPU {local_rank}...")
114     model = model.to(f"cuda:{local_rank}")
115
116     if world_size > 1:
117         print("Wrapping model in DistributedDataParallel (DDP
118             )...")
119         model = DDP(model, device_ids=[local_rank],
120             output_device=local_rank, find_unused_parameters=
121             False)
122
123     return model, tokenizer
124
125 def evaluate_model(model, tokenizer, dataset, max_samples=10)
126 :
127     metric = evaluate.load("rouge")
128     model.eval()
129     max_new_tokens = 1024
130
131     inputs = dataset["article"][:max_samples]
132     references = dataset["summary"][:max_samples]
133     predictions = []
134
135     with torch.no_grad():
136         for input_text in inputs:
137             inputs_tokenized = tokenizer(input_text,
138                 return_tensors="pt", truncation=True,
139                 max_length=16384,
140
141                                     padding="max_length"
142                                     )
143             inputs_tokenized = {k: v.to(model.device) for k,
144                 v in inputs_tokenized.items()}
145             outputs = model.generate(**inputs_tokenized,
146                 max_new_tokens=max_new_tokens, num_beams=2,
147                 early_stopping=True)
148             predictions.append(tokenizer.decode(outputs[0],
149                 skip_special_tokens=True))

```



```

179         tokenized_summary = tokenizer(summary, truncation=
180             False)["input_ids"]
181
182         special_token_length = len(tokenizer("[ARTICLE] [
183             SUMMARY]", add_special_tokens=False)["input_ids"])
184
185         total_length = len(tokenized_article) + len(
186             tokenized_summary) + special_token_length
187
188         if total_length > max_length:
189             max_article_tokens = max_length - len(
190                 tokenized_summary) - special_token_length
191             if max_article_tokens <= 0:
192                 continue
193
194             tokenized_article = tokenized_article[:
195                 max_article_tokens]
196
197             truncated_article = tokenizer.decode(
198                 tokenized_article)
199             truncated_summary = tokenizer.decode(
200                 tokenized_summary)
201
202             filtered_articles.append(truncated_article)
203             filtered_summaries.append(truncated_summary)
204
205         combined_inputs = [f"[ARTICLE] {article} [SUMMARY] {
206             summary}" for article, summary in
207             zip(filtered_articles,
208                 filtered_summaries)]
209
210         model_inputs = tokenizer(
211             combined_inputs,
212             truncation=True,
213             max_length=16384,
214             padding="max_length",
215             return_tensors=None
216         )
217
218         input_ids = model_inputs["input_ids"]
219         model_inputs["labels"] = model_inputs["input_ids"]
220         summary_token_id = tokenizer("[SUMMARY]",
221             add_special_tokens=False)["input_ids"][0]

```

```
212     summary_start_positions = [ids.index(summary_token_id)
213                                for ids in input_ids]
214
215     labels = [
216         [-100] * pos + ids[pos:]
217         for pos, ids in zip(summary_start_positions,
218                             input_ids)
219     ]
220
221     model_inputs["labels"] = labels
222     return model_inputs
223
224 def main():
225     local_rank, world_size = setup_distributed()
226     model, tokenizer = load_model_and_tokenizer(local_rank,
227                                                  world_size)
228
229     print("Loading dataset...")
230     dataset = load_dataset("pszemraj/
231                            scientific_lay_summarisation-plos-norm")
232
233     max_article_tokens = 16384
234
235     subset_train = dataset["train"]
236     subset_validation = dataset["validation"]
237     subset_test = dataset["test"]
238
239     print("Processing datasets...")
240     tokenized_train = subset_train.map(
241         lambda x: preprocess_function(x, tokenizer),
242         batched=True,
243         remove_columns=subset_train.column_names,
244         batch_size=4
245     )
246     tokenized_val = subset_validation.map(
247         lambda x: preprocess_function(x, tokenizer),
248         batched=True,
249         remove_columns=subset_validation.column_names,
250         batch_size=4
251     )
252
253     print("Setting up trainer...")
```

```
251     training_args = get_training_args(local_rank)
252     data_collator = DataCollatorForLanguageModeling(tokenizer
253                                                       =tokenizer, mlm=False)
254
255     trainer = GradientCheckpointingTrainer(
256         model=model,
257         args=training_args,
258         train_dataset=tokenized_train,
259         eval_dataset=tokenized_val,
260         tokenizer=tokenizer,
261         data_collator=data_collator,
262         callbacks=[LossLoggerCallback()]
263     )
264
265     print("Starting training...")
266     trainer.train()
267
268     if local_rank in [-1, 0]:
269         print("Saving model...")
270         unwrapped_model = model.module if hasattr(model, "module")
271         else model
272         unwrapped_model.save_pretrained("./mistral-instruct-
273                                         summarization-final")
274         tokenizer.save_pretrained("./mistral-instruct-
275                                   summarization-final")
276
277     print("Evaluating model after training...")
278
279     if __name__ == "__main__":
280         main()
```


A.5. Fine-tuning evaluation script

```

1 from datasets import load_dataset
2 from huggingface_hub import login
3 from transformers import AutoTokenizer, AutoModelForCausalLM
4 from peft import PeftModel
5 import torch
6 import evaluate
7 import sys
8
9 sys.stdout = open('finetuning_eval_log.txt', 'w')
10 sys.stderr = open('finetuning_eval_log.txt', 'w')
11
12 login(token="YOURTOKEN")
13
14
15 def load_test_data():
16     print("Loading test dataset...")
17     dataset = load_dataset("pszemraj/
18         scientific_lay_summarisation-plos-norm", split="test")
19
20     tokenizer = AutoTokenizer.from_pretrained("./instruct-
21         summarization-final", use_fast=True)
22
23     filtered_data = []
24     for sample in dataset:
25         article = sample["article"]
26         if len(tokenizer(article)["input_ids"]) <= 10000:
27             filtered_data.append(sample)
28         if len(filtered_data) >= 400:
29             break
30
31     print(f"Filtered dataset size: {len(filtered_data)}")
32     return filtered_data
33
34 def generate_summary(model, tokenizer, article,
35     max_input_tokens=10000, max_output_tokens=674):
36     input_text = f"[ARTICLE] {article} [SUMMARY]"
37     inputs = tokenizer(input_text, return_tensors="pt",
38         truncation=True, max_length=max_input_tokens).to("cuda")

```

```
37     with torch.no_grad():
38         output = model.generate(
39             **inputs,
40             max_new_tokens=max_output_tokens,
41             early_stopping=True,
42         )
43
44     decoded_output = tokenizer.decode(output[0],
45                                       skip_special_tokens=True)
46
47     if "[SUMMARY]" in decoded_output:
48         decoded_output = decoded_output.split("[SUMMARY]")
49         [-1].strip()
50
51     return decoded_output
52
53 def evaluate_model(model, tokenizer, dataset):
54     metric = evaluate.load("rouge")
55
56     references = []
57     predictions = []
58
59     for sample in dataset:
60         article, summary = sample["article"], sample["summary"]
61
62         generated_summary = generate_summary(model, tokenizer,
63                                             , article)
64
65         references.append(summary)
66         predictions.append(generated_summary)
67
68     results = metric.compute(predictions=predictions,
69                             references=references)
70
71     return results
72
73 def evaluate_base_model(test_data):
74     print("Loading base model...")
75     tokenizer = AutoTokenizer.from_pretrained("mistralai/
76                                             Mistral-7B-Instruct-v0.2", use_fast=True)
77     base_model = AutoModelForCausalLM.from_pretrained(
```

```
74         "mistralai/Mistral-7B-Instruct-v0.2",
75         torch_dtype=torch.bfloat16
76     ).to("cuda")
77
78     results = evaluate_model(base_model, tokenizer, test_data
79                             )
80     print("Evaluation Results (base):", results)
81
82     return results
83
84 def evaluate_finetuned_model(test_data):
85     print("Loading fine-tuned model with LoRa adapters...")
86     tokenizer = AutoTokenizer.from_pretrained("./instruct-
87         summarization-final", use_fast=True)
88     base_model = AutoModelForCausalLM.from_pretrained("
89         mistralai/Mistral-7B-Instruct-v0.2").to("cuda")
90
91     model = PeftModel.from_pretrained(base_model, "./instruct
92         -summarization-final").to("cuda")
93
94     results = evaluate_model(model, tokenizer, test_data)
95     print("Evaluation Results (fine):", results)
96     return results
97
98 if __name__ == "__main__":
99     test_data = load_test_data()
100
101     print("\nEvaluating Base Model:")
102     evaluate_base_model(test_data)
103
104     print("\nEvaluating Fine-tuned Model:")
105     evaluate_finetuned_model(test_data)
```