CCA – COMPETENCE CENTRE
HTL Anichstraße

Diplomarbeit

Artificial Intelligence in the Industry and Education Environment

SUBTITITLE

Eingereicht von

Gabriel Mrkonja
Florian Prandstetter
Luna Schätzle

Eingereicht bei

Höhere Technische Bundeslehr- und Versuchsanstalt Anichstraße

Abteilung für Wirtschaftsingenieure/Betriebsinformatik

Betreuer

Greinöcker
Egger

Projektpartner

HTL Anichstraße

Innsbruck, April 2025

Abgabevermerk:

Betreuer/in:

Datum:

SPERRVERMERK

Auf Wunsch der Firma

HTL Anichstraße

ist die vorliegende Diplomarbeit
für die Dauer von drei / fünf / sieben Jahren
für die öffentliche Nutzung zu sperren.
Veröffentlichung, Vervielfältigung und Einsichtnahme sind ohne
ausdrückliche Genehmigung der Firma *** und der Verfasser
bis zum TT.MM.JJJJ nicht gestattet.

Innsbruck, TT.MM.JJJJ

Verfasser:

Vor- und Zuname	Unterschrift
Vor- und Zuname	Unterschrift
Firma:	Firmenstempel

Kurzfassung /Abstract

Eine Kurzfassung ist in deutscher sowie ein Abstract in englischer Sprache mit je maximal einer A4-Seite zu erstellen. Die Beschreibung sollte wesentliche Aspekte des Projektes in technischer Hinsicht beschreiben. Die Zielgruppe der Kurzbeschreibung sind auch Nicht-Techniker! Viele Leser lesen oft nur diese Seite.

Beispiel für ein Abstract (DE und EN)

Die vorliegende Diplomarbeit beschäftigt sich mit verschiedenen Fragen des Lernens Erwachsener – mit dem Ziel, Lernkulturen zu beschreiben, die die Umsetzung des Konzeptes des Lebensbegleitenden Lernens (LBL) unterstützen. Die Lernfähigkeit Erwachsener und die unterschiedlichen Motive, die Erwachsene zum Lernen veranlassen, bilden den Ausgangspunkt dieser Arbeit. Die anschließende Auseinandersetzung mit Selbstgesteuertem Lernen, sowie den daraus resultierenden neuen Rollenzuschreibungen und Aufgaben, die sich bei dieser Form des Lernens für Lernende, Lehrende und Institutionen der Erwachsenenbildung ergeben, soll eine erste Möglichkeit aufzeigen, die zur Umsetzung dieses Konzeptes des LBL beiträgt. Darüber hinaus wird im Zusammenhang mit selbstgesteuerten Lernprozessen Erwachsener die Rolle der Informations- und Kommunikationstechnologien im Rahmen des LBL näher erläutert, denn die Eröffnung neuer Wege zur orts- und zeitunabhängiger Kommunikation und Kooperation der Lernenden untereinander sowie zwischen Lernenden und Lernberatern gewinnt immer mehr an Bedeutung. Abschließend wird das Thema der Sichtbarmachung, Bewertung und Anerkennung des informellen und nicht-formalen Lernens aufgegriffen und deren Beitrag zum LBL erörtert. Diese Arbeit soll einerseits einen Beitrag zur besseren Verbreitung der verschiedenen Lernkulturen leisten und andererseits einen Reflexionsprozess bei Erwachsenen, die sich lebensbegleitend weiterbilden, in Gang setzen und sie somit dabei unterstützen, eine für sie geeignete Lernkultur zu finden.

This thesis deals with the various questions concerning learning for adults – with the aim to describe learning cultures which support the concept of live-long learning (LLL). The learning ability of adults and the various motives which lead to adults learning are the starting point of this thesis. The following analysis on self-directed learning as well as the resulting new attribution of roles and tasks which arise for learners, trainers and institutions in adult education, shall demonstrate first possibilities to contribute to the implementation of the concept of LLL. In addition, the role of information and communication technologies in the framework of LLL will be closer described in context of self-directed learning processes of adults as the opening of new forms of communication and co-operation independent of location and time between learners as well as between learners and tutors gains more importance. Finally the topic of visualisation, validation and recognition of informal and non-formal learning and their contribution to LLL is discussed.

Gliederung des Abstract in **Thema**, **Ausgangspunk**, **Kurzbeschreibung**, **Zielsetzung**.

Projektergebnis Allgemeine Beschreibung, was vom Projektziel umgesetzt wurde, in einigen kurzen Sätzen. Optional Hinweise auf Erweiterungen. Gut machen sich in diesem Kapitel auch Bilder vom Gerät (HW) bzw. Screenshots (SW). Liste aller im Pflichtenheft aufgeführten Anforderungen, die nur teilweise oder gar nicht umgesetzt wurden (mit Begründungen).

CCA - COMPETENCE CENTRE
HTL Anichstraße

Erklärung der Eigenständigkeit der Arbeit

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe. Meine Arbeit darf öffentlich zugänglich gemacht werden, wenn kein Sperrvermerk vorliegt.

Ort, Datum Verfasser 1

Ort, Datum Verfasser 1

Inhaltsverzeichnis

Teil I

Introduction

1 Einleitung

In der Einleitung wird erklärt, wieso man sich für dieses Thema entschieden hat. (Zielsetzung und Aufgabenstellung des Gesamtprojekts, fachliches und wirtschaftliches Umfeld)

1.1 Vertiefende Aufgabenstellung

1.1.1 Schüler*innen Name 1

1.1.2 Schüler*innen Name 2

1.2 Dokumentation der Arbeit

Es werden die Projektergebnisse dokumentiert

- Grundkonzept
- Theoretische Grundlagen
- Praktische Umsetzung
- Lösungsweg
- Alternativer Lösungsweg
- Ergebnisse inkl. Interpretation

Weitere Anregungen:

- Fertigungsunterlagen
- Testfälle (Messergebnisse. . .)
- Benutzerdokumentation
- Verwendete Technologien und Entwicklungswerkzeuge

2 Introduction: AI in the Industry and Education Environment

3 Conceptual Evolution and Rationale

This chapter provides a comprehensive analysis of the project's evolution. It delineates the progression from an initial theoretical proposal, through the Self-Sufficiency Raspberry Pi Project, to the final project concept. The discussion elucidates the key motivations, challenges, and pivotal decisions that have ultimately shaped the design.

3.1 Introduction

In the context of this Diploma Thesis, the project has undergone several conceptual transformations aimed at refining its scope, objectives, and implementation strategy. Documenting this evolution is essential for a holistic understanding of the development process, as it highlights the iterative nature of design and the interplay between feasibility, scalability, and the initial research goals. Key considerations included evaluating the project's feasibility, ensuring scalability, and aligning with the overarching research objectives. Additionally, a critical examination of technical challenges encountered during implementation informed subsequent refinements.

3.2 Timeline and Milestones

To provide a clear overview of the project's evolution, a timeline outlining the major milestones, decision points, and revisions is presented in the following chart.

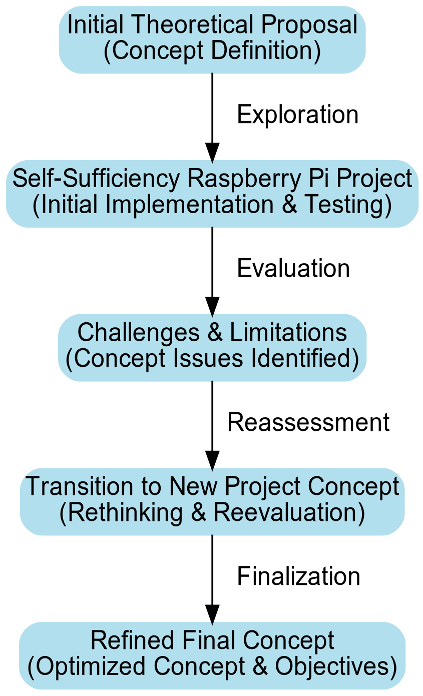
Initial Theoretical Proposal
(Concept Definition)
Exploration
Self-Sufficiency Raspberry Pi Project
(Initial Implementation & Testing)
Evaluation
Challenges & Limitations
(Concept Issues Identified)
Reassessment
Transition to New Project Concept
(Rethinking & Reevaluation)
Finalization
Refined Final Concept
(Optimized Concept & Objectives)

Abbildung 3.1: Gantt Chart of the Project Evolution

This timeline visually summarizes the progression of ideas and the key changes made over time, offering a concise reference to the project's developmental history.

3.3 Initial Concept

The first concept of the Diploma thesis was to investigate the different methodes how you can leverage AI in different fields and also to come up with a way to measure the qualety of the AI model for different tasks and use cases. The Idea was also to implement questonere to get a better understanding of the different use cases and how the AI model can be used in different fields. The goal was to come up with a way to measure the qualety of the AI model and to provide a way to compare different models with each other.

But the project team quickly realized that this concept was too broad and that it would be difficult to implement the project in the given time frame.

It is also not very clear how the project should be implemented and how the different parts of the project should be connected to each other.

3.4 The Self-Sufficiency Project

After the initial concept was deemed too broad and complex, the project team decided to focus on a more specific and tangible project idea. The Self-Sufficiency Project was born out of the need to create a practical and achievable project that could be implemented within the given time frame.

The idea was to develop a self-sufficient system based on a Raspberry Pi that could perform various tasks that would implement the use of AI models in a practical way. The self-sufficient part was about trying to eliminate the need of apis and other external services to make the system more robust and independent.

Another key factor of the project idea was to put the whole system in a cariable case so that it could be easily transported and used in different locations.

For a better understanding of the project, the following figure illustrates the conceptual idea of the Self-Sufficiency Rassberry Pi Project.

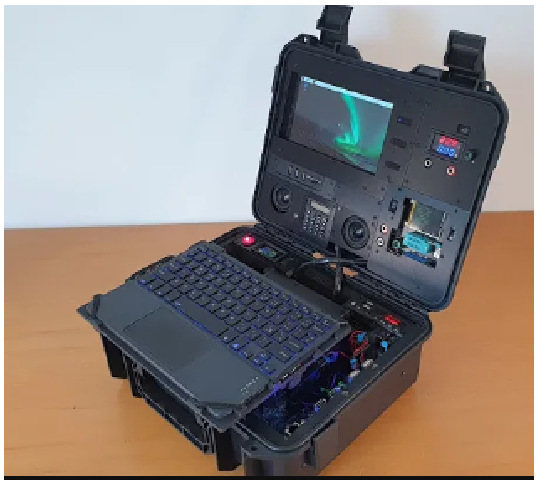
Abbildung 3.2: Conceptual Framework of the Self-Sufficiency Project

3.5 Challenges and Limitations of the Self-Sufficiency Project

As the project progressed, several challenges and limitations became apparent, prompting a reevaluation of the project's direction. These challenges included:

- **Complexity of Implementation:** The integration of various AI models and hardware components posed significant technical challenges, requiring extensive development and testing.
- **Scalability Concerns:** The self-sufficiency concept limited the project's scalability and interoperability with external systems, potentially hindering future expansion.
- **Resource Constraints:** The project's ambitious scope and resource requirements exceeded the available time and expertise, necessitating a more focused approach.
- **Lack of Clear Objectives:** The project lacked clear, measurable objectives and success criteria, making it challenging to assess progress and outcomes effectively.

- **Integration Complexity:** The integration of AI models, hardware components, and software systems proved more complex than anticipated, requiring a more streamlined approach.
- **Time consernes:** The project team quickly realized that the project would take more time than expected and that it would be difficult to implement the project in the given time frame.

Despite its initial promise, the Self-Sufficiency Project encountered several challenges. This section critically examines the practical issues and theoretical shortcomings that led to the reconsideration of the project direction.

3.6 Transition to the Current Project Concept

Based on the analysis of previous limitations, a new project concept was developed. This section explains the rationale behind the transition, details the improvements made, and describes the refined structure of the current project. It also outlines how the different components of the thesis address the project objectives.

3.7 Overcoming Challenges in the Current Project Concept

Every project evolution comes with its own set of challenges. In this section, we identify specific problems encountered in the current project concept and describe the strategies and solutions implemented to resolve them, ensuring the robustness of the final product.

3.8 Insights and Lessons Learned

Reflecting on the entire evolution process, this section summarizes the key insights gained and lessons learned. These reflections serve as guidance for future projects and contribute to a deeper understanding of the iterative design process.

3.9 Conclusion

The chapter concludes by summarizing the journey from the initial concept to the final project idea. It reiterates the importance of adaptive planning and critical analysis in the development process and highlights the contributions of each phase to the overall success of the project.

Teil II

Hardware

4 Raspberry PI

5 Server

6 Server Structure

Teil III

Theoretical background

7 Operating Systems used

8 Used Programming Languages

8.1 Python

To enhance readability and comprehension, the most widely used libraries are explained in the following sections.

8.1.1 MediaPipe

8.1.2 Transformers

The `Transformers` library, developed by Hugging Face, is an open-source Python package that provides implementations of state-of-the-art transformer models. It supports multiple deep learning frameworks, including PyTorch, TensorFlow, and JAX, facilitating seamless integration across various platforms. The library offers access to a vast array of pre-trained models tailored for tasks such as natural language processing, computer vision, and audio analysis. Utilizing these pre-trained models enables researchers and practitioners to achieve high performance in tasks like text classification, named entity recognition, and question answering without the necessity of training models from scratch, thereby conserving computational resources and time **?**.

8.1.3 JSON

JavaScript Object Notation (JSON) is a lightweight, text-based data interchange format that is easy for humans to read and write, and straightforward for machines to parse and generate. In Python, the built-in `json` module provides functionalities to serialize Python objects into JSON-formatted strings and deserialize JSON strings back into Python objects. This module supports the conversion of fundamental Python data types, such as dictionaries, lists, strings, integers, and floats, into their corresponding JSON representations. The `json` module is indispensable for tasks involving data exchange between Python applications and external systems, particularly in web development contexts where JSON is a prevalent format for client-server communication **?**.

Gabriels Part

8.2 HTML, CSS, and JavaScript in Combination with Vue.js

In this project, the languages HTML, CSS, and JavaScript were used in conjunction with Vue.js to create an interactive and dynamic user experience. For more information about Vue.js, see Chapter **??**, Section "Vue.js."

8.2.1 HyperText Markup Language (HTML)

HyperText Markup Language (HTML) is the standard markup language for creating and structuring content on the web. It serves as the backbone of web pages by organizing content through elements represented by tags. Key features of HTML include:

- **Structure Definition:** Tags such as `<html>`, `<head>`, and `<body>` define the structural hierarchy of a web page.
- **Content Organization:** Elements like headings, paragraphs, links, images, and tables provide a clear and user-friendly layout.
- **Web Compatibility:** HTML is universally supported, ensuring seamless integration across browsers and devices.

As one of the core technologies of the World Wide Web, alongside CSS and JavaScript, HTML enables the creation of interactive and visually appealing websites. Its simplicity and adaptability make it an essential tool for web development.

?

8.2.2 Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) is a style sheet language designed to control the visual presentation of web pages. CSS enhances the user experience by allowing developers to define the look and feel of a website. Key functionalities of CSS include:

- **Design Customization:** Control over layout, colors, fonts, and spacing for a cohesive visual identity.
- **Responsive Design:** Ensures consistent and optimized appearance across different devices and screen sizes.
- **Cascading Rules:** Allows styles to be applied at element, class, or global levels, offering flexibility in design.

As a foundational technology of the web, CSS plays a vital role in creating modern, responsive, and aesthetically pleasing websites.

?

8.2.3 JavaScript

JavaScript is a high-level programming language used to add interactivity and dynamic content to web pages. It works seamlessly alongside HTML and CSS to create rich and engaging user experiences. Key features of JavaScript include:

- **Dynamic Content:** Enables animations, form validation, and real-time updates.

- **Client and Server-Side Usage:** Runs in web browsers via JavaScript engines and supports server-side applications through platforms like Node.js.
- **Extensive Ecosystem:** Offers libraries, frameworks, and tools for building feature-rich web applications.

JavaScript's flexibility and versatility have established it as a cornerstone of web development, making it essential for developing interactive and responsive applications.

?

8.2.4 Vue.js

write about that VueJS is and that it uses those languages

8.3 Type Script

TypeScript is a superset of JavaScript that adds static type definitions to the language. It is designed for the development of large-scale applications and transcompiles to JavaScript. JavaScript is not very strict when it comes to types. This can lead to issues during development. Typescript adds a type system to JavaScript, which can help to catch errors early in the development process.

?

8.3.1 Axios

Axios is the key library used to make HTTP request from the frontend to the backend. It is a promise-based HTTP client for the browser and Node.js. It is used to make asynchronous requests to a server, and it returns a promise that resolves with the response data.

?

Flos Part

Teil IV

Implementation of Large Language Models

9 Overview and Integration of Large Language Models

For the Diploma thesis, there are many different AI models that are in use. There are different Types of AI models, such as:

- LLMs (Large Language Models)
- Defusion Models (Models that are used to create images)
- Object Detection Models (Models that are used to detect objects in images)
- Face Recognition Models (Models that are used to recognize faces in images)

In the following chapters, the different Types and the used models will be explained in more detail.

9.1 Large Language Models (LLMs)

Large Language Models (LLMs) represent a significant advancement in artificial intelligence, enabling machines to process and generate natural language. LLMs are built on the concept of deep learning, utilizing neural networks with billions of parameters to understand and generate text in a contextually accurate and coherent manner. These models are trained on vast datasets encompassing diverse topics, allowing them to handle a wide range of tasks, such as translation, summarization, content generation, and conversational AI.

9.1.1 Key Characteristics of LLMs

- **Scale and Complexity:** LLMs are distinguished by their immense size, often containing billions of parameters, enabling them to capture intricate patterns in language.
- **Transfer Learning:** These models benefit from pretraining on large datasets, followed by fine-tuning for specific tasks, making them highly versatile.
- **Contextual Understanding:** LLMs excel at understanding context, which allows them to generate coherent and contextually appropriate responses.
- **Multilingual Capabilities:** Many LLMs are trained on datasets in multiple languages, enabling them to process and generate text in various languages.

9.1.2 Applications of LLMs

- Text summarization and paraphrasing.
- Question answering and information retrieval.
- Conversational agents and chatbots.
- Code generation and debugging assistance.
- Creative writing, including story and poetry generation.

9.1.3 Examples of Popular LLMs

- **GPT Models:** Developed by OpenAI, these models include GPT-3, GPT-4, and ChatGPT, known for their state-of-the-art performance in text generation and comprehension.
- **BERT (Bidirectional Encoder Representations from Transformers):** Developed by Google, BERT focuses on understanding context by analyzing text bidirectionally.
- **LLama Models:** Created by Meta, these models are designed for efficient natural language understanding and generation.
- **Mistral Models:** Aimed at specialized tasks with high precision and multilingual capabilities.

9.1.4 Advantages and Challenges of LLMs

Advantages:

- High accuracy in generating and understanding text.
- Adaptability to a variety of domains and languages.
- Ability to process complex and context-rich queries.

Challenges:

- High computational and memory requirements.
- Potential biases due to the training data.
- Difficulty in maintaining factual accuracy in generated content.

?

9.2 Utilized Large Language Models

In the context of this diploma thesis, various free and commercial large language models (LLMs) were evaluated to determine their suitability for integration. Leveraging the Ollama application, we were able to test and compare several LLMs. Additionally, we explored different ChatGPT models available through the OpenAI API. OpenAI offers a range of models that vary in terms of size and complexity, with more advanced models incurring higher usage costs.

? ?

9.3 Ollama Application Overview

The Ollama application is an advanced, locally hosted platform designed to provide a versatile environment for deploying and interacting with a wide array of artificial intelligence models. It offers a comprehensive solution for both text and image processing tasks, facilitating the integration, fine-tuning, and management of models in a secure and scalable manner.

9.3.1 Ollama Features

Ollama is distinguished by several key features that enhance its functionality and usability:

- **Multi-Model Support:** The platform supports a variety of AI models, each optimized for specific tasks such as natural language processing and image analysis.
- **Local API Hosting:** The API is hosted on a local server, ensuring rapid and secure processing of requests while maintaining full control over data.
- **Image Processing Capabilities:** In addition to textual data, certain models within Ollama are capable of processing images. These models can analyze visual content, thereby extending the application's utility.
- **Model Customization and Fine-Tuning:** Users can fine-tune existing models to suit their specific needs. Once customized, these models can be re-uploaded to the Ollama server, allowing for continuous improvement and adaptation.

9.3.2 Ollama Architecture

The architecture of Ollama is modular and designed to support high performance and scalability:

1. **Model Management Layer:** This layer is responsible for deploying, fine-tuning, and updating the various AI models. It provides a structured approach to manage model versions and customizations.
2. **API Service Layer:** Hosted locally, this layer facilitates communication between client applications and the AI models. It exposes endpoints for both text and image processing, ensuring secure and efficient data exchange.
3. **Integration Interfaces:** These interfaces enable seamless connectivity with external services and applications, promoting interoperability and flexibility in diverse operational environments.

This layered design supports efficient resource management while enabling rapid response times and scalability to handle increasing user demands.

9.3.3 Ollama Models

Ollama provides a diverse selection of models, each tailored to specific application domains:

- **Text Generation Models:** Optimized for tasks such as dialogue generation, summarization, and other natural language processing applications.
- **Image Analysis Models:** Developed for image recognition, generation, and related tasks.

Furthermore, the platform allows users to fine-tune these models based on their particular requirements. Customized models can be re-uploaded to the server, enabling a continuous cycle of refinement and performance enhancement.

9.3.4 Ollama API

The Ollama API is the primary interface through which client applications interact with the hosted models. It provides robust and secure endpoints for processing both textual and visual data:

- **Data Exchange:** The API facilitates structured data exchange between client applications and the backend, ensuring that requests and responses are handled efficiently.
- **Security and Performance:** Designed with stringent security protocols, the API ensures that all interactions are encrypted and managed in a way that maximizes performance while minimizing latency.
- **Extensibility:** The API's modular design allows for the easy addition of new endpoints and functionalities as the platform evolves.

9.3.5 Ollama Integration

Integrating the Ollama API into external applications is straightforward. For instance, a Python-based client can send HTTP requests to the API to perform tasks such as generating text or processing images. This section is further elaborated in the chapter dedicated to the hosted Flask Service, where detailed examples and implementation guidelines are provided. In brief, the integration involves:

- Establishing a connection to the local API endpoint.
- Sending appropriately formatted requests (e.g., JSON payloads) that include user inputs.
- Handling responses from the API, which may include generated text or URLs to processed images.

9.3.6 Benefits and Challenges of Ollama

Ollama presents several benefits:

- **Ease of Use:** The platform is user-friendly, with intuitive APIs that simplify deployment and integration.
- **Versatility:** A wide array of models enables the application of Ollama to diverse tasks, from natural language processing to image analysis.
- **Multilingual Support:** The models are capable of processing multiple languages, thereby broadening the scope of potential applications.
- **Customization:** Users can fine-tune models to meet specific needs and update them on the server, ensuring tailored performance.

However, several challenges must be addressed:

- **Performance Limitations:** Larger models may experience slower response times due to higher computational demands.
- **API Request Management:** Ensuring that the API can handle a high volume of requests efficiently requires robust load balancing and error handling mechanisms.
- **Model Management Complexity:** Coordinating updates, fine-tuning, and deployment of multiple models demands an effective management strategy.
- **Concurrency:** Managing simultaneous user requests, as discussed in the chapter on the hosted Flask Service, is critical to maintaining system performance under high load.

In summary, while Ollama offers a flexible and powerful platform for AI model deployment and interaction, addressing its inherent challenges is crucial for optimizing performance and ensuring long-term scalability in practical applications.

?

9.4 Evaluation of Models via the Ollama Platform

In this project, we conducted an evaluation of various models accessible through the Ollama application, which are available for download from the Ollama server.

Given that Ollama operates locally, it was imperative to select models that align with specific criteria to ensure optimal performance. Consequently, we assessed models of diverse sizes and complexities to determine their suitability for local deployment. This evaluation encompassed both the efficacy and efficiency of the models within a local environment.

?

9.4.1 Model Selection Criteria

The selection of models was guided by the following criteria:

- **Model Size:** The model must be capable of running on the server without exceeding available memory capacity.
- **Performance Speed:** The response time of the model, i.e., how quickly it can generate output.
- **Complexity:** The model's ability to handle complex prompts and generate coherent, contextually accurate text.
- **Accuracy:** The overall precision of the model's responses, particularly in terms of factual correctness and linguistic quality.
- **Language Support:** The model's proficiency in understanding and generating text in multiple languages, particularly English and German.

- **User Experience:** The model's overall usability and user-friendliness, including ease of integration and customization.

There is often a trade-off between these criteria. Larger models tend to exhibit higher accuracy and greater contextual understanding but are generally slower and require more computational resources.

9.4.2 Challenges in Model Testing and Updates

One significant challenge lies in the rapid development and frequent release of new models, which complicates the process of continuous integration and comprehensive evaluation of recent advancements. Regular testing and updates are imperative to ensure the incorporation of state-of-the-art models while maintaining system reliability and relevance.

Another challenge involves achieving an optimal balance between performance, accuracy, and resource efficiency, ensuring that the chosen model meets the application's functional requirements without compromising the overall user experience.

During the evaluation process, we encountered several obstacles. For instance, identifying standardized questions that could be uniformly answered by all models proved challenging. Some smaller models demonstrated limitations in addressing certain questions comprehensively. Additionally, specialized models, while excelling in niche areas, often lacked the ability to provide detailed answers across a broader range of topics.

For the final evaluation phase, we employed a diverse question set consisting of self-crafted questions, publicly available questions from online sources, and queries generated by ChatGPT-4 to ensure a comprehensive assessment covering a wide spectrum of queries.

9.4.3 Model Selection for the Final Application

In the final implementation, users are provided with a curated list of recommended models from which they can select their preferred option. This list was carefully compiled based on our comprehensive testing and reflects the models that demonstrated the best balance between performance, accuracy, and resource efficiency.

For the production version of the application, this list must be updated periodically to include newly released models and maintain optimal performance.

9.5 Ollama Model Testing and Evaluation

Based on the following criteria, we conducted an extensive evaluation of the upcomming models:

9.5.1 Quantitative Evaluation Methods

For the quantitative evaluation, we focused on key performance metrics to assess the efficiency and reliability of each model:

- **Response Time:** The time taken by the model to generate a response after receiving input.
- **CPU Usage:** The percentage of CPU resources utilized during model execution.
- **GPU Usage:** The extent to which GPU resources were leveraged to enhance performance.
- **Memory Usage:** The amount of RAM consumed while the model was running.
- **Multiple Choice Question Answering:** The accuracy of the model when answering structured multiple-choice questions.

9.5.2 Qualitative Evaluation Methods

While qualitative evaluation is inherently resource-intensive due to its reliance on human judgment, it remains essential for assessing aspects that cannot be fully captured through quantitative metrics. Consequently, although our primary focus was on quantitative evaluation, we conducted qualitative assessments for key criteria where human input was indispensable:

- **Translation Quality:** Evaluated using the BLEU (Bilingual Evaluation Understudy) score, which measures the similarity between the model-generated translation and a human reference translation. **?**
- **Text Generation Quality:** Assessed through the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) score, which quantifies the lexical overlap between generated text and reference texts.
- **Grammatical Accuracy:** Manually reviewed by human evaluators to identify grammatical errors and assess syntactic correctness.
- **Readability:** Measured using the Flesch Reading Ease score, which indicates the complexity and accessibility of the generated text.
- **Sentiment Polarity:** Analyzed to determine whether the generated text conveys a positive, negative, or neutral sentiment.

?

By integrating both quantitative and qualitative evaluation methods, we achieved a more comprehensive understanding of each model's strengths and weaknesses, allowing for a well-rounded assessment of their performance.

9.5.3 Models Evaluated During Testing

For the evaluation process, we selected some of the most popular models available in the Ollama application and conducted extensive testing on each. The models vary in size, specialization, and intended use cases, covering general-purpose, coding, mathematical, reasoning, and image-processing tasks.

- **qwen2.5-coder:0.5b** – A compact model with 0.5 billion parameters, specifically designed for coding-related tasks.
- **qwen2.5-coder:7b** – A small-scale model with 7 billion parameters, optimized for software development and code generation.

- **qwen2.5-coder:14b** – A mid-sized model with 14 billion parameters, tailored for complex coding tasks.7
- **qwen2-math** – A specialized model with 1 billion parameters, fine-tuned for mathematical computations.
- **llama3.2:1b** – A lightweight model with 1 billion parameters, designed by Meta for general-purpose applications.
- **llama3.2:2b** – A medium-sized model with 2 billion parameters, developed by Meta for a broader range of general-purpose tasks.
- **mistral:7b** – A versatile 7-billion-parameter model created by Mistral AI, a European AI company, for general applications.
- **mathstral** – A specialized model optimized for mathematical problem-solving, developed by Mistral AI.
- **phi4:14b** – A mid-sized model with 14 billion parameters, designed by Microsoft for general-purpose reasoning tasks.
- **deepseek-r1:1.5b** – A small-scale model with 1.5 billion parameters, featuring enhanced reasoning capabilities.
- **deepseek-r1:7b** – A 7-billion-parameter model optimized for reasoning tasks, offering a balance between performance and hardware compatibility.
- **deepseek-r1:14b** – A more powerful variant with 14 billion parameters, fine-tuned for complex reasoning tasks.
- **gemma2** – A lightweight model with 1 billion parameters, designed by Google for general-purpose applications.
- **llava:13b** – A large-scale model with 13 billion parameters, developed for image processing tasks by a dedicated research team. **?**

?

9.5.4 Data Collection

For the Data collection we used the School AI Server wich was provided by the HTL Anichstraße, to run the different models in the Evaluation phase. For the later Data collection we used our own Server to run the different models.

To colletct the data we used a varaitey of different python scripts. For the quantitative data we used the following python script:

```python
import time
import json
import psutil  # For CPU, memory usage
from ollama import chat

# Prompts for testing
prompts = [
    "Explain the theory of relativity in simple terms.",
    "Create a short story about a knight.",
    "What are the advantages of open-source projects?",
    "Write a Python function that outputs prime numbers up to 100.",
    # ...
]

# Model name
model_name = "qwen2-math"

# Store results
results = []

# Function to get GPU usage if available
def get_gpu_usage():
    try:
        import torch
        if torch.cuda.is_available():
            gpu_memory = torch.cuda.memory_allocated() / (1024 ** 2)  # Convert to MB
            gpu_utilization = torch.cuda.utilization(0) if hasattr(torch.cuda, 'utilization') else "N/A"
            return gpu_memory, gpu_utilization
        else:
            return 0, "No GPU detected"
    except ImportError:
        return 0, "torch not installed"

# Loop through prompts
for prompt in prompts:
    try:
        # Measure system usage before model execution
        cpu_before = psutil.cpu_percent(interval=None)
        memory_before = psutil.virtual_memory().used / (1024 ** 2)  # Convert to MB

        start_time = time.time()
        # Ollama chat request
        response = chat(model=model_name, messages=[{'role': 'user', 'content': prompt}])
        end_time = time.time()

        latency = end_time - start_time

        # Measure system usage after model execution
        cpu_after = psutil.cpu_percent(interval=None)
        memory_after = psutil.virtual_memory().used / (1024 ** 2)  # Convert to MB

        cpu_usage = cpu_after - cpu_before
        memory_usage = memory_after - memory_before
        gpu_memory_usage, gpu_utilization = get_gpu_usage()

        # Extract content from the Message object
        if response and hasattr(response["message"], "content"):
            response_text = response["message"].content  # Accessing the attribute of the Message object
        else:
            response_text = "No content returned or unexpected format"

        print(f"Prompt: {prompt}\nResponse Time: {latency:.2f} seconds\n")

        # Save the result
        results.append({
            "Prompt": prompt,
            "Response Time (seconds)": latency,
            "Response": response_text,
            "CPU Usage (%)": cpu_usage,
            "Memory Usage (MB)": memory_usage,
            "GPU Memory Usage (MB)": gpu_memory_usage,
                "GPU Utilization (%)": gpu_utilization
            })
        except Exception as e:
            print(f"Error with prompt '{prompt}': {e}")
            results.append({
                "Prompt": prompt,
                "Response Time (seconds)": "Error",
                "Response": f"Error: {str(e)}",
                "CPU Usage (%)": "N/A",
                "Memory Usage (MB)": "N/A",
                "GPU Memory Usage (MB)": "N/A",
                "GPU Utilization (%)": "N/A"
            })20.01.2025

# Save to JSON file
json_file_name = model_name + "_response_time_results_ressours_usage.json"
with open(json_file_name, "w") as file:
    json.dump(results, file, indent=4)

print(f"The results have been saved in {json_file_name}.")
```

Listing 9.1: Python-quantitative-data-collection

For this Pyhton Skript we used ChatGPT to help with the psutil library to get the CPU and Memory usage. We also used ChatGPT to get more Questions for the data collection.

The results were saved as JSON files for the later analysis. One Problem we encountered were the tourch librarys, witch diden't function probally on the School AI Server, so we couldn't get the GPU usage for the models.

Used python libaries

psutil libarie

The `psutil` library is a Python module that provides an interface for retrieving information on system utilization, including CPU, memory, disk, network, and processes. It is commonly used for monitoring and managing system performance and is highly efficient due to its low overhead. `psutil` is cross-platform, supporting major operating systems like Windows, Linux, and macOS. It enables developers to create scripts for system diagnostics, process control, and resource management, making it an essential tool for performance optimization and system administration in Python-based projects.

?

Ollama

The `ollama` library is a Python package designed to provide a seamless interface for interacting with the Ollama application. It allows users to easily access and leverage various AI models for natural language processing tasks. By simplifying the integration of AI models into Python applications, the library supports a wide range of functionalities, making it an efficient tool for developing AI-powered solutions.

?

9.5.5 Data Preperation

To get more information about the collected data, we used the following Python script to collect more data:

```
import json
import os
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from rouge_score import rouge_scorer
import language_tool_python
import textstat
from transformers import pipeline, logging

# Suppress warning messages from the Transformer library for a cleaner output.
logging.set_verbosity_error()

def load_json(file_path):
    """
    Load and parse JSON data from a file.

    Parameters:
        file_path (str): The file system path to the JSON file.

    Returns:
        dict or list: The JSON data parsed from the file.
    """
    with open(file_path, ’r’) as file:
        return json.load(file)

def calculate_metrics(data):
    """
    Compute multiple evaluation metrics for generated text responses.

    For each data item, the function calculates:
    - BLEU Score: Quantifies the similarity between the generated response and the reference text.
    - ROUGE Scores: Evaluates the n-gram overlap between the reference and the generated text, using ROUGE-1, ROUGE-2, an
    - Grammar Check: Determines the number of grammatical errors present in the response.
    - Readability Score: Computes the Flesch Reading Ease score to assess the text’s readability.
    - Sentiment Analysis: Infers the sentiment polarity (e.g., positive or negative) of the response text.

    Parameters:
        data (list): A list of dictionaries, each containing ’Prompt’, ’Response’, and ’Reference’ keys.

    Returns:
        list: A list of dictionaries that include the original text elements along with the computed metrics.
    """
    results = []
    rouge = rouge_scorer.RougeScorer([’rouge1’, ’rouge2’, ’rougeL’], use_stemmer=True)
    grammar_tool = language_tool_python.LanguageTool(’en-US’)
    sentiment_analyzer = pipeline(
        ’sentiment-analysis’,
        model="distilbert-base-uncased-finetuned-sst-2-english",
            truncation=True,
            max_length=512
        )

        for item in data:
            prompt = item['Prompt']
            response = item['Response']
            reference = item['Reference']

            try:
                # Calculate the BLEU Score with smoothing to address potential issues with short sequences.
                bleu_score = sentence_bleu(
                    [reference.split()], response.split(),
                    smoothing_function=SmoothingFunction().method4
                )

                # Calculate ROUGE Scores for comprehensive n-gram overlap assessment.
                rouge_scores = rouge.score(reference, response)

                # Perform grammatical analysis by counting detected errors in the response.
                grammar_errors = len(grammar_tool.check(response))

                # Determine the readability score using the Flesch Reading Ease metric.
                readability_score = textstat.flesch_reading_ease(response)

                # Analyze the sentiment of the response text, with error handling to capture any exceptions.
                try:
                    sentiment_result = sentiment_analyzer(response)[0]
                    sentiment = sentiment_result['label']
                except Exception as e:
                    print(f"Sentiment error for prompt '{prompt}': {e}")
                    sentiment = "Error"

                results.append({
                    "Prompt": prompt,
                    "Response": response,
                    "Reference": reference,
                    "BLEU": bleu_score,
                    "ROUGE-1": rouge_scores['rouge1'].fmeasure,
                    "ROUGE-2": rouge_scores['rouge2'].fmeasure,
                    "ROUGE-L": rouge_scores['rougeL'].fmeasure,
                    "Grammar Errors": grammar_errors,
                    "Readability Score": readability_score,
                    "Sentiment": sentiment
                })
            except Exception as e:
                print(f"Error processing prompt '{prompt}': {e}")
                results.append({
                    "Prompt": prompt,
                    "Response": response,
                    "Reference": reference,
                    "Error": str(e)
                })

        return results

def save_results(results, output_path):
    """
    Persist the computed metrics to a JSON file.

    Parameters:
        results (list): A list of dictionaries containing evaluation metrics and corresponding texts.
        output_path (str): The file system path where the output JSON should be saved.
    """
    with open(output_path, 'w') as file:
        json.dump(results, file, indent=4)

def main():
    """
    Execute the main workflow of the script.

    This function prompts the user to specify a directory containing preprocessed JSON files.
    It then iterates through each file that matches the pattern 'processed_*.json',
    computes the evaluation metrics for the contained data,
    and saves the results in a new JSON file prefixed with 'scored_'.
    """
    directory = input("Enter the directory containing the processed JSON files: ")

    try:
        for file_name in os.listdir(directory):
            if file_name.startswith("processed_") and file_name.endswith(".json"):
                input_file = os.path.join(directory, file_name)
                model_name = file_name.split("processed_")[1].split(".json")[0]
                output_file = os.path.join(directory, f"scored_{model_name}.json")

                print(f"Processing file: {input_file}")
                data = load_json(input_file)
                metrics = calculate_metrics(data)
                save_results(metrics, output_file)
                print(f"Metrics saved to: {output_file}")

    except FileNotFoundError:
        print(f"Error: The directory {directory} was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    main()
```

Listing 9.2: Python-data-preperation-for-analysis

This Python script implements a comprehensive evaluation framework for assessing the quality of generated textual responses. It systematically processes JSON files containing a prompt, a generated response, and a reference text, computing several quantitative metrics: the BLEU score for assessing n-gram overlap, ROUGE metrics for evaluating text similarity, a grammatical error count via LanguageTool, the Flesch Reading Ease score for readability, and sentiment analysis using a Transformer-based model. By integrating these diverse analytical techniques from state-of-the-art natural language processing libraries, the script facilitates a rigorous and multifaceted scientific evaluation of text generation performance.

Utilized Python Libraries

For Collecting those data we used the following Python Libraries:

Natural Language Toolkit (NLTK) The Natural Language Toolkit (NLTK) is a comprehensive library for natural language processing in Python. It provides easy-to-use interfaces to over 50 corpora and lexical resources, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. NLTK is widely used for building Python programs that work with human language data and is a leading platform in both research and education **?**.

ROUGE Score The `rouge-score` library is a native Python implementation of the ROUGE metric, which is commonly used for evaluating automatic summarization and machine translation. It replicates the results of the original Perl package and supports the computation of ROUGE-N (N-gram) and ROUGE-L (Longest Common Subsequence) scores. The library also offers functionalities such as text normalization and the use of stemming to enhance evaluation accuracy **?**.

LanguageTool Python `language-tool-python` is a wrapper for LanguageTool, an open-source grammar, style, and spell checker. This library enables the integration of LanguageTool's proofreading capabilities into Python applications, supporting the detection and correction of grammatical errors, stylistic issues, and spelling mistakes across multiple languages.

Textstat The `textstat` library provides simple methods for calculating readability statistics from text. It helps determine the readability, complexity, and grade level of textual content by computing various metrics such as the Flesch Reading Ease, SMOG Index, and Gunning Fog Index. These metrics are valuable for assessing and ensuring the comprehensibility of text, particularly in educational and professional settings **?**.

9.5.6 Data Processing

To gain a better understanding of the collected data, we utilized Python scripts to generate visualizations, providing a clearer representation of the results. Additionally, the processed data was formatted into a LaTeX table to facilitate structured analysis and comparison.

Quantitative Data Analysis

To visualize the quantitative data, we employed the following Python script:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import glob
import numpy as np
import os
import logging

# Set up logging for consistent error and information messages.
logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

# Set scientific plotting style with an increased default figure height.
plt.style.use('default')
sns.set_theme(style="whitegrid", context="paper")
plt.rcParams.update({
    'font.family': 'serif',
    'font.serif': ['Times New Roman'],
    'font.size': 10,
    'axes.labelsize': 12,
    'axes.titlesize': 14,
    'xtick.labelsize': 10,
    'ytick.labelsize': 10,
    'figure.dpi': 300,
    'savefig.dpi': 300,
    'figure.figsize': (8, 10),
    'grid.color': '#e0e0e0',
    'axes.spines.right': False,
    'axes.spines.top': False
})

def load_and_process_data() -> pd.DataFrame:
    """
    Loads and processes all JSON files in the current directory.

    This function searches for all files matching "*.json", reads them into pandas DataFrames,
    assigns a 'Model' column based on the file name (without extension), converts specified
    numeric columns to numeric type, and removes rows with missing values in those columns.

    Returns:
        pd.DataFrame: A concatenated and cleaned DataFrame containing all data.
    """
    json_files = glob.glob("*.json")
    if not json_files:
        logging.warning("No JSON files found in the current directory.")
        return pd.DataFrame()

    dfs = []
    for file in json_files:
        try:
            model_name = os.path.splitext(file)[0]
            df = pd.read_json(file)
            df['Model'] = model_name
            dfs.append(df)
        except Exception as e:
            logging.error(f"Error loading {file}: {e}")

    if not dfs:
        logging.error("No data could be loaded from the JSON files.")
        return pd.DataFrame()

    combined_df = pd.concat(dfs, ignore_index=True)

    # Convert selected columns to numeric and drop rows with missing values in these columns.
    numeric_cols = ['Response Time (seconds)', 'CPU Usage (%)', 'Memory Usage (MB)']
    combined_df[numeric_cols] = combined_df[numeric_cols].apply(pd.to_numeric, errors='coerce')
    combined_df = combined_df.dropna(subset=numeric_cols)

    return combined_df

def create_resource_plot(df: pd.DataFrame, metric: str, title: str, ylabel: str, filename: str) -> None:
    """
    Creates a resource usage plot with violin and strip plots, annotated with statistical measures.

    The function generates a violin plot for the given metric across different AI models, overlays a strip plot
    to display individual data points, and annotates each model with its median and mean absolute deviation (MAD).
    The resulting plot is saved in both PDF and PNG formats.

    Parameters:
        df (pd.DataFrame): DataFrame containing the metric and 'Model' columns.
        metric (str): The column name representing the metric to be visualized.
        title (str): The title of the plot.
        ylabel (str): The label for the y-axis.
        filename (str): Base filename used for saving the plot.
    """
    plt.figure(figsize=(10, 10))

    ax = sns.violinplot(
        x='Model',
        y=metric,
        data=df,
        inner='quartile',
        palette='muted',
        cut=0
    )

    sns.stripplot(
        x='Model',
        y=metric,
        data=df,
        color='#303030',
        size=2.5,
        alpha=0.7
    )

    # Calculate median and mean absolute deviation (MAD) for each model.
    stats = df.groupby('Model')[metric].agg(median='median', mad=lambda x: np.mean(np.abs(x - x.median())))

    # Annotate each model with the calculated median and MAD.
    for xtick, model in enumerate(stats.index):
        model_stats = stats.loc[model]
        annotation = f"Med: {model_stats['median']:.1f}\nMAD: {model_stats['mad']:.1f}"
        # Position annotation at 5% above the minimum value.
        y_pos = df[metric].min() + (df[metric].max() - df[metric].min()) * 0.05
        ax.text(
            xtick,
            y_pos,
            annotation,
            ha='center',
            va='bottom',
            fontsize=8,
            color='#404040'
        )

    plt.title(title, pad=15)
    plt.xlabel('AI Model', labelpad=12)
    plt.ylabel(ylabel, labelpad=12)
    plt.xticks(rotation=45, ha='right')
    plt.ylim(bottom=0)
    plt.tight_layout()

    # Save the plot in both vector (PDF) and raster (PNG) formats.
    plt.savefig(f'{filename}.pdf', bbox_inches='tight')
    plt.savefig(f'{filename}.png', bbox_inches='tight')
    plt.close()

def plot_cpu_memory_comparison(df: pd.DataFrame) -> None:
    """
    Generates comparative plots for CPU usage and memory consumption across AI models.

    This function calls 'create_resource_plot' for both CPU and Memory metrics.

    Parameters:
        df (pd.DataFrame): DataFrame containing performance metrics.
    """
    create_resource_plot(
        df=df,
        metric='CPU Usage (%)',
        title='Comparative Analysis of CPU Utilization Across AI Models',
        ylabel='CPU Usage (%)',
        filename='model_cpu_usage_comparison'
        )

        create_resource_plot(
            df=df,
            metric='Memory Usage (MB)',
            title='Comparative Analysis of Memory Consumption Across AI Models',
            ylabel='Memory Usage (MB)',
            filename='model_memory_usage_comparison'
        )

    def generate_advanced_statistics(df: pd.DataFrame) -> None:
        """
        Generates advanced performance statistics for AI models and outputs the results both in the console and as a LaTeX t

        The statistics include mean, standard deviation, and maximum values for CPU and memory usage,
        as well as mean and standard deviation for response times.

        Parameters:
            df (pd.DataFrame): DataFrame containing performance metrics.
        """
        stats = df.groupby('Model').agg({
            'CPU Usage (%)': ['mean', 'std', 'max'],
            'Memory Usage (MB)': ['mean', 'std', 'max'],
            'Response Time (seconds)': ['mean', 'std']
        })

        print("\nAdvanced Performance Statistics:")
        print(stats.round(2).to_string())

        # Export the statistics as a formatted LaTeX table.
        try:
            latex_str = stats.style.format({
                ('CPU Usage (%)', 'mean'): "{:.1f}",
                ('Memory Usage (MB)', 'mean'): "{:.1f}"
            }).to_latex(
                hrules=True,
                caption="Model Performance Statistics",
                label="tab:model_stats"
            )
            with open('resource_stats.tex', 'w') as f:
                f.write(latex_str)
        except Exception as e:
            logging.error(f"Error generating LaTeX table: {e}")

    def plot_response_times(df: pd.DataFrame) -> None:
        """
        Creates a comparative boxplot for model response times overlaid with a swarm plot for individual data points.

        The function annotates each AI model with its median response time and saves the plot in both PDF and PNG formats.

        Parameters:
            df (pd.DataFrame): DataFrame containing the 'Response Time (seconds)' and 'Model' columns.
        """
        plt.figure(figsize=(8, 10))

        ax = sns.boxplot(
            x='Model',
            y='Response Time (seconds)',
            data=df,
            width=0.6,
            showfliers=False,
            palette='muted'
        )

        sns.swarmplot(
            x='Model',
            y='Response Time (seconds)',
            data=df,
            color='#404040',
            size=3,
            alpha=0.6
        )

        # Annotate the median response time for each model.
        medians = df.groupby('Model')['Response Time (seconds)'].median()
```

```python
    for xtick, model in enumerate(medians.index):
        median_val = medians.loc[model]
        ax.text(
            xtick,
            median_val + 0.05,
            f'{median_val:.2f}s',
            ha='center',
            va='bottom',
            fontsize=8,
            color='#2f2f2f'
        )

    plt.title('Comparative Analysis of Model Response Times', pad=15)
    plt.xlabel('AI Model', labelpad=10)
    plt.ylabel('Response Time (seconds)', labelpad=10)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()

    plt.savefig('model_response_times_comparison.pdf', bbox_inches='tight')
    plt.savefig('model_response_times_comparison.png', bbox_inches='tight')
    plt.close()

def generate_statistics(df: pd.DataFrame) -> None:
    """
    Generates a statistical summary of response times for each AI model and exports the results.

    The summary includes the mean, standard deviation, minimum, median, and maximum values.
    The results are printed to the console and saved as a LaTeX table.

    Parameters:
        df (pd.DataFrame): DataFrame containing the 'Response Time (seconds)' and 'Model' columns.
    """
    stats = df.groupby('Model')['Response Time (seconds)'].describe()
    print("\nResponse Time Statistics:")
    print(stats[['mean', 'std', 'min', '50%', 'max']].round(3).to_string())

    try:
        with open('response_stats.tex', 'w') as f:
            f.write(
                stats[['mean', 'std', 'min', '50%', 'max']]
                .round(3)
                .style.to_latex(hrules=True)
            )
    except Exception as e:
        logging.error(f"Error generating LaTeX response stats: {e}")

def main() -> None:
    """
    Main execution function.

    Loads and processes data from JSON files, generates various comparative plots (response times, CPU, and memory usage),
    and outputs advanced performance statistics along with their LaTeX representations.
    """
    df = load_and_process_data()
    if df.empty:
        logging.error("No data available for plotting and analysis.")
        return

    plot_response_times(df)
    plot_cpu_memory_comparison(df)
    generate_advanced_statistics(df)
    generate_statistics(df)

if __name__ == "__main__":
    main()
```

Listing 9.3: Python-quantitative-data-analysis

This script performs a comprehensive performance evaluation of various AI models by loading JSON files from the working directory, extracting key metrics such as response time, CPU usage, and memory consumption, and preprocessing the data for analysis.

It generates high-resolution visualizations—including violin, strip, box, and swarm plots—to effectively illustrate the distributions and central tendencies of these metrics. Additionally, it computes descriptive statistics and presents the results both in the console and as LaTeX-formatted tables, ensuring structured and reproducible scientific reporting.

Qualitative Data Analysis

To visualize the qualitative data, we utilized the following Python script:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

# ============================================================================
# Data Aggregation and Visualization for Model Performance Metrics
# ============================================================================
# This script aggregates experimental results from JSON files, each containing
# performance metrics (e.g., BLEU, ROUGE, grammatical errors, readability, sentiment)
# for various AI models. The data are visualized using high-quality plots for scientific
# analysis, and descriptive statistics are exported in LaTeX format.

# ----------------------------
# Data Aggregation
# ----------------------------
directory = "./"
aggregated_data = []
for file in os.listdir(directory):
    # Process files that follow the naming convention "scored_<model>.json"
    if file.startswith("scored_") and file.endswith(".json"):
        model = file.replace("scored_", "").replace(".json", "")
        df_temp = pd.read_json(os.path.join(directory, file))
        df_temp["Model"] = model
        aggregated_data.append(df_temp)
df = pd.concat(aggregated_data, ignore_index=True)

# ----------------------------
# Global Plotting Style Settings
# ----------------------------
sns.set_theme(style="whitegrid", font_scale=0.9)
plt.rcParams['axes.titlepad'] = 15
plt.rcParams['axes.labelpad'] = 10

def rotate_labels(ax, rotation: int = 45, ha: str = 'right') -> None:
    """
    Rotate the x-axis labels for improved readability.

    Parameters:
        ax (matplotlib.axes.Axes): The axes on which to rotate the labels.
        rotation (int): Angle in degrees to rotate the labels.
        ha (str): Horizontal alignment of the labels.
    """
    ax.set_xticklabels(ax.get_xticklabels(), rotation=rotation, ha=ha, fontsize=9)
    plt.tight_layout()

# ============================================================================
# Visualization of Performance Metrics
# ============================================================================

# --- 1. BLEU and ROUGE Scores ---
plt.figure(figsize=(14, 7))
```

```python
# Reshape data for plotting multiple text quality metrics
df_melt = df.melt(id_vars=["Model"], value_vars=["BLEU", "ROUGE-1", "ROUGE-2", "ROUGE-L"],
                  var_name="Metric", value_name="Score")
ax = sns.barplot(x="Model", y="Score", hue="Metric", data=df_melt, palette="viridis")
plt.title("Comparison of BLEU and ROUGE Scores", fontweight='bold')
plt.ylim(0, 0.05)
plt.legend(loc="upper right", frameon=True)
rotate_labels(ax)
plt.savefig("bleu_rouge.png", dpi=300, bbox_inches="tight")

# --- 2. Grammatical Errors ---
plt.figure(figsize=(12, 6))
ax = sns.boxplot(x="Model", y="Grammar Errors", data=df, palette="Set2")
plt.title("Distribution of Grammatical Errors per Model", fontweight='bold')
rotate_labels(ax)
plt.savefig("grammar_errors.png", dpi=300, bbox_inches="tight")

# --- 3. Readability (Flesch Score) ---
plt.figure(figsize=(12, 6))
ax = sns.pointplot(x="Model", y="Readability Score", data=df, capsize=0.1, palette="coolwarm")
plt.title("Average Readability (Flesch Score)", fontweight='bold')
rotate_labels(ax)
plt.savefig("readability.png", dpi=300, bbox_inches="tight")

# --- 4. Sentiment Analysis ---
# Compute relative frequency of sentiment labels per model
sentiment_counts = df.groupby(["Model", "Sentiment"]).size().unstack().fillna(0)
sentiment_percent = sentiment_counts.div(sentiment_counts.sum(axis=1), axis=0) * 100

plt.figure(figsize=(14, 7))
ax = sentiment_percent.plot(kind="bar", stacked=True, colormap="RdYlGn")
plt.title("Sentiment Distribution of Responses", fontweight='bold')
plt.ylabel("Percentage (%)")
plt.legend(title="Sentiment", bbox_to_anchor=(1.05, 1))
rotate_labels(ax)
plt.savefig("sentiment.png", dpi=300, bbox_inches="tight")

# --- 5. Combined Metrics Overview ---
fig, axes = plt.subplots(2, 2, figsize=(18, 14))

# Subplot 1: BLEU & ROUGE Metrics
sns.barplot(ax=axes[0, 0], x="Model", y="Score", hue="Metric", data=df_melt)
axes[0, 0].set_title("Text Quality (BLEU & ROUGE)", fontweight='bold')
rotate_labels(axes[0, 0])

# Subplot 2: Grammatical Errors
sns.boxplot(ax=axes[0, 1], x="Model", y="Grammar Errors", data=df, palette="Set2")
axes[0, 1].set_title("Grammatical Errors", fontweight='bold')
rotate_labels(axes[0, 1])

# Subplot 3: Readability
sns.pointplot(ax=axes[1, 0], x="Model", y="Readability Score", data=df, capsize=0.1, palette="coolwarm")
axes[1, 0].set_title("Readability", fontweight='bold')
rotate_labels(axes[1, 0])

# Subplot 4: Sentiment
sentiment_percent.plot(ax=axes[1, 1], kind="bar", stacked=True, colormap="RdYlGn")
axes[1, 1].set_title("Sentiment", fontweight='bold')
rotate_labels(axes[1, 1])

plt.subplots_adjust(hspace=0.3, wspace=0.25)
plt.savefig("combined_metrics.png", dpi=300, bbox_inches="tight")

# ==========================================================================
# Descriptive Statistics Export
# ==========================================================================
# Compute summary statistics for selected metrics by model
summary = df.groupby("Model")[["BLEU", "ROUGE-L", "Grammar Errors"]].agg(["mean", "std", "median", "min", "max"])
summary.to_latex("summary.tex", float_format="%.3f")
```

Listing 9.4: Python-qualitative-data-analysis

This script aggregates performance metrics from multiple JSON files—each corresponding to an AI model evaluation—into a unified dataset. It then generates high-resolution visualizations, including bar, box, and point plots, to illustrate text quality (BLEU/ROUGE scores), grammatical accuracy, readability, and sentiment distribution. Finally, it computes descriptive statistics for these metrics and exports a summary table in LaTeX format for rigorous scientific reporting.

Utilized Python Libraries

In this project, several Python libraries were employed to facilitate data manipulation, analysis, and visualization:

Pandas Pandas is an open-source data analysis and manipulation library for Python. It provides data structures such as Series and DataFrames, which allow for efficient handling of structured data. Pandas supports operations like data alignment, merging, and reshaping, making it indispensable for data preprocessing and analysis tasks.

?

Matplotlib Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It offers an object-oriented API for embedding plots into applications and supports various plot types, including line, bar, scatter, and 3D plots. Matplotlib's flexibility and extensive customization options make it a fundamental tool for data visualization.

?

Seaborn Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics, including functions for visualizing univariate and bivariate distributions, categorical data, and linear regression models. Seaborn integrates closely with Pandas data structures, enhancing the aesthetic appeal and interpretability of visualizations.

?

NumPy NumPy is a foundational library for numerical computing in Python. It introduces support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy serves as the backbone for many other libraries, including Pandas and Matplotlib, by providing efficient array operations and numerical computations.

?

9.5.7 Testresults and Analysis

Write about the Analysis and include the images of the graphics

9.5.8 Model Comparison for different Use Cases and Scenarios

Explain wich modell is best for what

9.5.9 Model Selected for the Final Application

Explain why the selected model was selected and what the criteria were

and explain the different models.

9.5.10 Model Integration and Deployment

Following the evaluation process, the selected models were systematically integrated into the School AI Server and the Visual Studio Code extension.

Leveraging the user-friendly API provided by the Ollama application, we facilitated a seamless integration of the models into the School AI Server, ensuring efficient accessibility and deployment. To enhance request management and optimize communication between different components, we developed a Python-based Flask server that hosts a dedicated API. This API serves as an intermediary layer, enabling structured and scalable interactions between the School AI Server and the Visual Studio Code extension.

A comprehensive discussion of the hosted Flask service, including its architecture and functionality, is presented in Chapter 10: *Hosted Flask Service*.

9.6 Integration of OpenAI's API

In this work, we integrated the OpenAI API to leverage proprietary, high-performance AI models that are hosted on dedicated servers with advanced hardware capabilities. The utilization of external computing power allows for the concurrent execution of multiple models, thereby enhancing both scalability and efficiency in our application.

The decision to adopt the OpenAI API was influenced by its widespread adoption, robust performance, and extensive documentation. Numerous examples, tutorials, and community resources are available, which greatly facilitate the integration process and ensure that best practices are followed in scientific and industrial applications.

9.6.1 Overview of the OpenAI API

The OpenAI API provides access to state-of-the-art AI models developed by OpenAI, including various iterations of the ChatGPT model. These models are capable of generating human-like text, answering queries, and engaging in complex conversations. The API supports a range of models with different sizes and capabilities, allowing users to select the model that best fits the requirements of their specific use cases.

Designed with user accessibility in mind, the API comes with comprehensive documentation and a wealth of code samples, which significantly streamline the process of embedding advanced AI functionalities into diverse applications and platforms. Furthermore, the API utilizes a token-based pricing model, which charges users according to the number of tokens processed during interactions. This pricing structure is not only transparent but also aligns closely with the computational effort required to generate responses.

Before accessing the API's full functionality, users must pre-fund their accounts by depositing a specified amount of money. This account-based billing system enables users to manage their expenditures effectively, including the option to set monthly spending limits. In addition to text generation, the OpenAI ecosystem also includes DAL-E, an image-generation model that creates visuals based on textual input, thus broadening the spectrum of applications available through the API.

?

Tokens in Large Language Models

Tokens are the fundamental units of text that large language models (LLMs) process and generate. In this context, a token represents the smallest segment of text that a model can understand, which may correspond to an entire word, a fragment of a word, or even an individual character or punctuation mark.

The process of tokenization involves converting raw text into these discrete units. This approach enables LLMs to efficiently capture complex patterns in both syntax and semantics, even when encountering new or out-of-vocabulary terms. Techniques such as subword tokenization are particularly valuable, as they break down words into meaningful components, thereby reducing the overall vocabulary size and enhancing the model's ability to manage linguistic variability.

Moreover, tokens are closely related to the concept of a context window, which defines the span of tokens a model can consider during text generation or prediction. Typically, one token is estimated to average around four characters in English or roughly three-quarters of a word. This estimation is crucial for determining computational requirements and understanding the limitations imposed by the model's finite context window.

In summary, tokens are indispensable for the operation of LLMs, providing a structured means to process language. Their effective management through advanced tokenization strategies is essential for optimizing both the computational efficiency and the overall performance of these models.

?

9.6.2 Data Security and Privacy in Compliance with Austrian and EU Regulations

The integration of OpenAI's API into our systems necessitates a thorough examination of data security and privacy considerations, particularly in the context of Austrian and European Union (EU) regulations. The General Data Protection Regulation (GDPR) serves as the cornerstone of data protection within the EU, imposing stringent requirements on the processing of personal data.

OpenAI has implemented several measures to safeguard user data and align with GDPR mandates. Notably, they support compliance with privacy laws such as the GDPR and the California Consumer Privacy Act (CCPA), offering a Data Processing Addendum to customers. Their API and related products have undergone evaluation by an independent third-party auditor, confirming alignment with industry standards for security and confidentiality.

?

Despite these measures, concerns have been raised regarding data handling practices. For instance, data transmitted through the OpenAI API could potentially be exposed, and compliance with GDPR remains a complex issue. Additionally, data may be accessible to third-party subprocessors, introducing further privacy considerations.

?

To address these concerns, we have proactively informed our user community through a notice on the school website. This notice outlines the data handling practices associated with the OpenAI API and provides guidance on how users can manage their data when interacting with our systems. By maintaining transparency and offering clear instructions, we aim to uphold the highest standards of data security and privacy in our academic environment.

In light of the evolving regulatory landscape, it is imperative to remain vigilant and responsive to any changes in data protection laws within Austria and the broader EU. Continuous monitoring and adaptation of our data handling practices will ensure ongoing compliance and the safeguarding of user privacy.

9.6.3 OpenAI API Implementation in Vue.js

This section details the integration of the OpenAI API within a Vue.js application framework, with a focus on both text and image generation capabilities. The implementation not only illustrates the interaction between the Vue.js frontend and the OpenAI API but also demonstrates adherence to security best practices and modular code design. The following discussion is supported by annotated code examples and an explanation of the libraries used.

Overview of the Implementation

The implementation is structured as a Vue.js component that facilitates the following functionalities:

- Accepting user input via a text area.
- Initiating API calls for generating text responses (using ChatGPT models) and creating images (via the DALL-E endpoint).

- Displaying the results (generated text and images) dynamically within the user interface.

The component is designed with a clear separation between presentation and business logic, ensuring that the code remains both maintainable and scalable.

Explanation of the Used Libraries

OpenAI Library The `openai` library is employed as the primary interface to interact with OpenAI's API endpoints. This library abstracts the complexities of HTTP communication and provides a user-friendly API to access advanced AI functionalities such as natural language generation and image synthesis. Its integration simplifies the process of constructing API requests and handling responses, which is critical for developing robust AI-driven applications.

API Key Management To ensure secure handling of sensitive credentials, the OpenAI API key is imported from an external module (i.e., `OPENAI_API_KEY` from the `secrets` file). This approach adheres to security best practices by preventing the direct embedding of API keys within the source code, thereby mitigating the risk of unauthorized exposure.

Code Example: Vue.js Component for OpenAI API Integration

Below is an illustrative example of a Vue.js component that integrates the OpenAI API for both text and image generation. The code is presented in two parts: the HTML template and the JavaScript logic.

Listing 9.5: Vue.js Template for OpenAI API Integration

```
<template>
  <div class="openai-container">
    <h1>OpenAI API Integration in Vue.js</h1>
    <textarea
```

```
      v-model="userInput"
      placeholder="Enter␣your␣prompt␣here..."
      rows="4"
      cols="50">
    </textarea>
    <div class="action-buttons">
      <button @click="generateText">Generate Text</button>
      <button @click="generateImage">Generate Image</button>
    </div>
    <div v-if="generatedText" class="output-section">
      <h2>Generated Text</h2>
      <p>{{ generatedText }}</p>
    </div>
    <div v-if="generatedImage" class="output-section">
      <h2>Generated Image</h2>
      <img :src="generatedImage" alt="Image␣generated␣by␣OpenAI␣API" />
    </div>
  </div>
</template>
```

Listing 9.6: Vue.js Script for OpenAI API Integration

```
<script>
import OpenAI from "openai";
import { OPENAI_API_KEY } from "../secrets";

export default {
  name: "OpenAIComponent",
  data() {
    return {
      userInput: "",
      generatedText: "",
      generatedImage: ""
    };
  },
  methods: {
    async generateText() {
      // Initialize OpenAI client with API key
      const openai = new OpenAI({ apiKey: OPENAI_API_KEY });
      try {
        const response = await openai.chat.completions.create({
          model: "gpt-3.5-turbo",
          messages: [{ role: "user", content: this.userInput }]
        });
        // Extract and assign the generated text
        this.generatedText = response.choices[0].message.content;
      } catch (error) {
        console.error("Error during text generation:", error);
      }
    },
    async generateImage() {
      // Initialize OpenAI client for image generation
      const openai = new OpenAI({ apiKey: OPENAI_API_KEY });
      try {
        const response = await openai.images.generate({
          prompt: this.userInput,
          n: 1,
          size: "512x512"
        });
        // Extract and assign the URL of the generated image
        this.generatedImage = response.data[0].url;
      } catch (error) {
        console.error("Error during image generation:", error);
      }
    }
  }
};
</script>
```

Discussion

The presented component exemplifies how modern web applications can seamlessly integrate AI capabilities while maintaining a secure and modular architecture. Key points of consideration include:

- **Modularity:** The separation of the UI (HTML template) and the business logic (JavaScript methods) facilitates easier maintenance and potential scalability.
- **Security:** By importing the API key from an external secrets module, the risk of credential leakage is minimized. This practice is crucial in academic and production environments where data security is paramount.
- **Extensibility:** The design allows for further expansion, such as additional error handling mechanisms or the integration of more advanced functionalities provided by the OpenAI API.

In conclusion, this integration not only demonstrates the practical application of AI APIs in modern web development but also reflects best practices in secure and maintainable code design. Such an approach is essential for building reliable applications in both academic research and industrial contexts.

9.7 Conclusion

10 hosted Flask Service

This chapter delineates the implementation, architecture, and deployment of the self-hosted Flask service, which functions as a pivotal interface between the front-end and back-end components of the system. In addition to detailing the technical design, the chapter critically examines the advantages of a self-hosted solution and the rationale behind key architectural decisions.

10.1 Introduction

This section provides a comprehensive overview of the motivations for developing the Flask service. It discusses the specific requirements that necessitated the service, outlines its core functionalities, and situates the service within the broader system architecture.

10.2 Advantages of a Self-hosted Service

In this section, the benefits of deploying a self-hosted service are explored in depth. The discussion includes:

- Enhanced customization and control over the service environment.
- The ability to rapidly prototype and deploy bespoke functionalities.
- Improved data security and compliance with institutional policies.

These factors collectively underscore the strategic choice of a self-hosted approach over third-party solutions.

10.3 Architecture and Service Structure

10.3.1 System Architecture

This subsection describes the monolithic architecture of the Python Flask server, elucidating how various components are integrated to form a cohesive whole. The internal workflow and interaction between modules are detailed to provide a clear picture of the service's operational logic.

10.3.2 Modularity and Extensibility

An emphasis is placed on the modular design of the service, which facilitates maintainability and scalability. This part discusses the design choices that allow for future enhancements and the integration of additional functionalities.

10.4 Flask as a Web Framework

10.4.1 Core Functionalities

This subsection explains the intrinsic capabilities of Flask, such as request routing, templating, and middleware support. It illustrates how these features are employed to manage web requests and responses within the service.

10.4.2 Rationale for Selecting Flask

A critical discussion is presented on why Flask was chosen for the project. Key factors include its simplicity, extensive documentation, and robust community support, which collectively make it an ideal framework for rapid development and prototyping.

10.5 RESTful Endpoints and Functionalities

10.5.1 Endpoint Specifications

This section enumerates the various RESTful endpoints implemented in the service. Each endpoint is described in detail, outlining its specific purpose and the type of data it handles.

10.5.2 Code Illustrations

To enhance understanding, code examples are provided to demonstrate the implementation of key endpoints. These examples highlight the methods used to process requests and generate responses.

10.5.3 Utility Functions

An overview of auxiliary utility functions is given, focusing on their roles in logging, data validation, and error handling. These functions contribute to the overall robustness and reliability of the service.

10.6 Utilized Libraries

A comprehensive inventory of the external libraries used in the project is presented in this section. For each library, its functionality, role within the project, and integration aspects are discussed.

10.7 Deployment

10.8 Docker

Docker is a platform that allows you to run applications in containers. A container is like a small, isolated environment where software runs with everything it needs – including the operating system, libraries, and dependencies.

No matter which computer or server the container runs on, it always works the same way. This means you don't have to worry about an application suddenly throwing errors on a different system just because a different software version is installed there.

Docker is often used in software development and cloud applications because it simplifies testing, deployment, and scaling of apps. Developers can store their software as images and share them with others without requiring complicated installations.

10.8.1 Used Docker Images

A docker image is a bluebrint that specifies how to run the application. The Instructions for the build are stored in the Dockerfile. **?**

- **flask**$_{a}$*ppTheflaskimageisusedtoeasilyimplementtheflaskapplicationinadockercontainer***olla**

10.8.2 Docker Compose

Docker Compose is used for running multiple containers at the same time. It simplifies your application and makes it easier to manage The Configuration is stored in a single YAML file. All the services can be started with a simpel command. It is a very compact way to manage Docker application. **?**

10.9 Conclusion and Future Work

This concluding section synthesizes the chapter's key points and reflects on the efficacy of the implemented service. It also outlines potential avenues for future enhancements, such as further scalability improvements, additional functionalities, and more robust deployment automation.

11 Studen AI Website

12 Visual Studio code extension

12.1 Introduction

This chapter provides an overview of the Visual Studio Code extension developed for the project. It describes its core functionalities, and explains how it integrates with the broader system architecture.

12.2 what is Visual Studio Code

Visual Studio Code is a free code editor from Microsoft. It supports many programming languages such as Python, JavaScript, and C++.

A major advantage of VS Code is its extensibility. With extensions, you can customize the editor, for example, with debugging tools, themes, or special functions for specific programming languages. It also offers features like auto-completion, integrated Git support, and a built-in terminal function.

VS Code is lightweight and runs on Windows, macOS, and Linux. Despite this, it provides many features that are also found in a full-fledged integrated development environment. This makes it perfect for both beginners and professionals.

12.3 Development

- TypeScript: The Visual Studio Code extension was developed using TypeScript. TypeScript is well-suited for developing VS Code exten-

sions, as it provides type checking and code completion, making it easier to work with the VS Code API.
- Axios: Axios is used to make HTTP requests from the extension to the Flask Service. It provides an easy implementation of asynchronous requests and simplifies handling responses.
- Visual Studio Code API: The extension interacts with the Visual Studio Code API. The API allows the extension to access and modify the editor's functionality, enabling it to provide a seamless development experience.

12.4 Core Functionalities

The planed core Functionalities of the extension are, an integrated chatbot and code completion.

12.4.1 Chatbot Integration

The extension integrates a chatbot into the editor, allowing developers to interact with the chatbot directly from the editor. This feature enables developers to quickly get information, ask questions, or perform tasks without leaving the editor.

12.4.2 Code completion

Teil V

Implementation of Object Detection

13 Introduction to Object Detection

14 Implementation of Object Detection

CCA – COMPETENCE CENTRE
HTL Anichstraße

Teil VI

Evaluations

15 Artificial Intelligence in Economics

15.1 Introduction

16 Open source evaluation on Economics

16.1 Introduction

This chapter introduces the concept of Open Source and highlights its significance in the modern economy. Key aspects such as the advantages and disadvantages of Open Source, as well as the challenges associated with its adoption and creation, are discussed. Additionally, the chapter explores revenue models within the Open Source ecosystem and its role in economic systems. Finally, the chapter concludes by presenting the Open Source tools utilized in this project, alongside a reflection on the experiences gained through their application.

16.1.1 What is Open Source?

Open Source represents a collaborative and transparent approach to software development and distribution, where the source code is made publicly accessible. This philosophy empowers users not only to utilize the software but also to modify, improve, and redistribute it freely. By fostering an environment of openness and collaboration, Open Source drives innovation and democratizes access to technology.

Linus Torvalds, the creator of the Linux operating system, encapsulated this spirit of freedom and collaboration with his famous remark:

> *"Software is like sex: it's better when it's free."*

?

This statement highlights the fundamental ethos of Open Source—the belief that open access and shared knowledge result in better, more impactful solutions.

The development process for Open Source software is often a collective effort, with contributions from diverse communities of developers, users, and organizations. These collaborative efforts enhance the software's functionality, security, and usability, resulting in products that are robust and adaptable. Prominent examples include the Linux operating system, the Apache web server, and the Firefox web browser, all of which have significantly influenced technological innovation and market dynamics.

?

16.1.2 Advantages of Open Source

Open Source software offers a wide range of benefits, making it a cornerstone of modern technology:

- **Cost Efficiency:** Open Source software is typically free of charge, helping organizations and individuals save on licensing and maintenance costs.
- **Flexibility:** Users can access the source code, enabling them to tailor the software to their specific needs and requirements.
- **Security:** The open nature of the source code allows for peer review, ensuring vulnerabilities are identified and addressed promptly.
- **Community Support:** Open Source projects often benefit from vibrant developer communities, providing updates, patches, and user assistance.
- **Innovation:** The collaborative ecosystem of Open Source encourages creativity, leading to groundbreaking solutions and advancements.
- **Compatibility:** Many Open Source projects are designed to integrate seamlessly with existing systems, reducing technical barriers.
- **Transparency:** Open access to the source code ensures that users can understand and verify how the software operates.

- **Freedom:** Users are granted the liberty to use, modify, and share the software without restrictive licensing agreements.

? ?

16.1.3 Why Do People Use Open Source?

The adoption of Open Source software is motivated by several compelling factors:

- **Control:** Users gain full control over the software, enabling customization and optimization for specific use cases.
- **Cost Savings:** The absence of licensing fees significantly reduces expenses, making Open Source particularly attractive for startups and educational institutions.
- **Security:** Transparency in the source code allows for thorough auditing, enhancing trust and reliability.
- **Community:** The collaborative spirit of Open Source connects users with knowledgeable communities that share resources and support.
- **Stability:** Many Open Source projects offer long-term support and regular updates, ensuring reliability over time.
- **Skill Development:** Learning and using Open Source tools are valuable in educational and professional contexts, equipping individuals with in-demand skills.

16.2 What is and isn't Open Source?

16.2.1 Definition and Guiding Principles

Open Source, as defined by the Open Source Initiative (OSI), is a development approach that prioritizes accessibility and transparency of software source code. It allows users to view, modify, and distribute the code freely, fostering collaboration and innovation.

The OSI outlines several key principles that define Open Source software:

- **Free Redistribution:** The software can be freely shared and distributed without restrictions.
- **Source Code Access:** Users must have access to the source code to study, modify, and improve the software.
- **Modification and Sharing:** Users are allowed to create and share modified versions, as long as they follow the license terms.
- **No Discrimination:** The software must be available for everyone, regardless of individual characteristics or professional field.
- **Neutrality and Compatibility:** The license must not favor specific technologies or restrict the use of other software.

These principles ensure that Open Source remains a transparent, inclusive, and adaptable approach to software development, enabling innovation and collaboration across industries and communities.

?

16.2.2 Misconceptions About Open Source

Open Source is often misunderstood and confused with other software distribution models, which can lead to misconceptions about its nature, functionality, and benefits. It is crucial to distinguish Open Source from other types of software:

- **Open Source:** Software that is freely accessible, modifiable, and redistributable under an Open Source license, adhering to principles such as transparency and collaboration.
- **Freeware:** Software available at no cost but typically without access to the source code, meaning users cannot modify or redistribute it.
- **Proprietary Software:** Software owned and controlled by a single entity, restricting access to the source code and preventing users from making modifications or redistributions.
- **Commercial Software:** Software sold for profit, which may be either Open Source or proprietary, depending on the licensing terms.

Understanding these distinctions helps users make informed choices about software selection and ensures their expectations align with the capabilities and freedoms provided by the chosen software.

To verify whether a software is truly Open Source, it is essential to examine the license agreement and confirm the availability of the source code. Software with an OSI-approved license is a reliable indicator that it adheres to Open Source principles, providing transparency, freedom, and collaboration opportunities.

One common misconception about Open Source software arises from the phrase "free as in freedom"versus "free as in free beer."While "free as in freedomëmphasizes the liberty to access, modify, and share the software, "free as in free beerßimply denotes that the software is free of cost. Although Open Source software is often available without charge, its true value lies in the freedom it grants to users, developers, and organizations. This distinction highlights the broader significance of Open Source as a philosophy, not just a pricing model.

?

16.3 The Role of Open Source in Economics

Cost efficiency, innovation, and collaboration are key factors that have positioned Open Source as a cornerstone of modern economic systems. Many industries and organizations utilize Open Source software to reduce costs, increase flexibility, and promote creativity, thereby driving economic growth and sustainability.

16.3.1 Driving Innovation and Shaping Market Dynamics

Open Source software fosters a culture of experimentation, creativity, and knowledge sharing, leading to the rapid development of new technologies and solutions. By granting users access to modify and redistribute the source code, Open Source encourages collaboration and innovation, enabling individuals and organizations to build upon existing software to create new products and services.

A distinctive strength of Open Source is its inclusivity—anyone, regardless of their affiliation with a company, can contribute to its development. This openness lowers barriers to entry for innovation and allows passionate individuals to make meaningful contributions.

Companies also play a significant role in advancing Open Source projects. With greater resources and structured teams, organizations can contribute in a more organized and impactful manner, accelerating development and enhancing software quality.

The collaborative nature of Open Source facilitates cross-industry partnerships, allowing organizations from diverse sectors to share knowledge, resources, and best practices. This cross-pollination of ideas not only enhances software development but also fosters innovation across industries, ultimately shaping market dynamics and driving economic progress.

The study **?** by Mike Hendrickson, Roger Magoulas, and Tim O'Reilly underscores that Open Source is not only a catalyst for small business growth but also a driver of future success for many startups today. By providing cost-effective and flexible solutions, Open Source enables small and medium-sized enterprises to strengthen their online presence and enhance their economic performance.

16.3.2 Supporting Startups and small Enterprises

The impact of Open Source on startups and small enterprises is both profound and transformative. For these businesses, Open Source software provides a highly cost-effective alternative to proprietary solutions, granting access to advanced tools and technologies without the financial burden of high licensing fees typically associated with commercial software. This affordability allows startups and small enterprises to allocate their limited resources more strategically, fostering innovation and growth while maintaining financial flexibility.

?

16.3.3 Enabling Cross-Industry Collaboration and Open Innovation

16.4 Advantages and Disadvantages of Open Source

16.4.1 Advantages

Open Source software offers numerous advantages for users, developers, and businesses. It can vary from cost savings to increased innovation and flexibility for customization.

- Cost savings.
- Flexibility for customization.
- Increased innovation due to open collaboration.

16.4.2 Disadvantages

- Reliance on community support.
- Potential security vulnerabilities.
- Compatibility issues with other systems.

16.5 Challenges of Using or Creating Open Source

There are many challanges that come with using or creating Open Source software. These can range from technical to economic and social challenges. Understanding these challenges is crucial for successful Open Source adoption and development.

16.5.1 Technical Challenges

- Maintaining quality and long-term compatibility.
- Managing security and privacy risks.

16.5.2 Economic Challenges

- Monetization and sustainability concerns.
- Balancing free access with profitability.

16.5.3 Social Challenges

- Effective community management and governance.

16.5.4 Legal Issues

- Navigating complex licensing models (e.g., GPL, MIT).

16.6 Revenue Models in Open Source

Open Source projects can generate revenue through various business models, each with its own advantages and challenges.

- Common business models:
  - Freemium.
  - Support and maintenance services.
  - Dual licensing.
  - Crowdfunding and donations.
- Real-world examples of successful Open Source businesses (e.g., Linux, Red Hat, MySQL).

16.7 Open Source in Key Industries

- The role of Open Source in transforming:
  - Information Technology (e.g., operating systems, tools).
  - Artificial Intelligence (e.g., TensorFlow, PyTorch).
  - Education (e.g., Moodle, Jupyter Notebooks).
- Governmental and policy support for Open Source adoption.

16.8 Reflexion

- Answering the research question based on the above analysis.
- Evaluating the broader implications of Open Source for economic systems.
- Connecting Open Source's potential with sustainability and global development.

16.9 Open Source in Practice: A Personal Experience

- Open Source tools and technologies used in the project:
  - Python, Flask, Vue.js, Linux, wttr.in API, LLaMA API.
- Challenges and solutions encountered:
  - Technical hurdles.
  - Why Open Source alternatives were chosen or rejected.
- Comparison of Open Source and closed-source software used:
  - Reasons for choosing closed-source alternatives where applicable.

16.10 Open Source in Our Project & Licensing

16.10.1 Project

- Description of the project.
- How Open Source principles were applied.
- Benefits and challenges of Open Source in the project.

16.10.2 License

- Choice of license and rationale.
- How the license aligns with the project's goals.
- The license problems of the project.
- Future plans for the project's development and licensing.

16.11 Conclusion

- Summary of Open Source's economic impact.
- Reflections on its potential to drive future innovation and growth.
- Final thoughts on your personal experience and insights gained.

17 Economic aspect of Operating Systems

17.1 Introduction

Teil VII

Conclusion

18 Proplems that occured

19 Conclusion

20 Outlook

Anhang A

Time Protocol

Make 3 Sections: For Every Person one Section with the Time Table What the Person Did when on the Project.

Appendix

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

9.1 Python-quantitative-data-collection 44
9.2 Python-data-preperation-for-analysis 46
9.3 Python-quantitative-data-analysis 50
9.4 Python-qualitative-data-analysis 54
9.5 Vue.js Template for OpenAI API Integration 62
9.6 Vue.js Script for OpenAI API Integration 63

Literaturverzeichnis

10 biggest advantages of open-source software (2022). [Online; accessed 2025-01-28].
URL: *https://www.rocket.chat/blog/open-source-software-advantages*

A Comprehensive Guide to Ollama - Cohorte Projects (n.d.). [Online; accessed 2025-02-14].
URL: *https://www.cohorte.co/blog/a-comprehensive-guide-to-ollama*

Ankush (2024), 'What is ollama? everything important you should know'. [Online; accessed 2025-01-13].
URL: *https://itsfoss.com/ollama/*

Axios Docs (2025). Accessed: 2025-02-14.
URL: *https://axios-http.com/docs/intro*

Bansal, S. and Aggarwal, C. (2025), 'Textstat: Python library for text statistics'. Zugriff am 13. Februar 2025.
URL: *https://pypi.org/project/textstat/*

Bird, S., Klein, E. and Loper, E. (2025), 'Natural language toolkit (nltk)'. Zugriff am 13. Februar 2025.
URL: *https://www.nltk.org/*

$Docker_{F}lask(n.d.).[Online;accessed2025-02-13].$
$URL:$*https://www.freecodecamp.org/news/how-to-dockerize-a-flask-app/*

$Doocker_{C}ompose(n.d.).[Online;accessed2025-02-13].$
$URL:$*https://docs.docker.com/compose/*

Forbes Technology Council (2024), 'Misconceptions about open source solutions clarified by tech experts'. Accessed: 2024-12-04.
URL: *https://www.forbes.com/councils/forbestechcouncil/2024/10/09/misconceptions-about-open-source-solutions-clarified-by-tech-experts/*

Fortis, S. (2024), 'Openai hit with privacy complaint in austria, potential eu law breach'. [Online; accessed 2025-02-07].
URL: *https://cointelegraph.com/news/openai-privacy-complaint-austria-potential-eu-law-breach*

Foundation, P. S. (2025), 'json — json encoder and decoder'. Accessed: 2025-02-14.
URL: *https://docs.python.org/3/library/json.html*

Foy, P. (2024), 'Understanding tokens & context windows'. [Online; accessed 2025-02-07].
URL: *https://blog.mlq.ai/tokens-context-window-llms/*

Hendrickson, M., Magoulas, R. and O'Reilly, T. (2012), *Economic Impact of Open Source on Small Business: A Case Study*, O'Reilly Media.
URL: *https://www.oreilly.com/library/view/economic-impact-of/9781449343408/*

HTML - Wikipedia (2001). [Online; accessed 2025-01-23].
URL: *https://en.wikipedia.org/wiki/HTML*

Hunter, J. D. and the Matplotlib Development Team (2025), 'Matplotlib: Python plotting library'. Accessed: 2025-02-14.
URL: *https://matplotlib.org/*

IBM (n.d.), 'What are large language models (llms)? | ibm'. [Online; accessed 2025-01-21].
URL: *https://www.ibm.com/think/topics/large-language-models*

Initiative, O. S. (2007), 'The open source definition', https://opensource.org/osd. Accessed: 2024-12-02.

Introducing data residency in Europe | OpenAI (n.d.). [Online; accessed 2025-02-07].
URL: *https://openai.com/index/introducing-data-residency-in-europe/?utm$_s$ource = chatgpt.com*

Liu, H., Li, C., Li, Y. and Lee, Y. J. (2023), 'Improved baselines with visual instruction tuning'.

McKinney, W. and the Pandas Development Team (2025), 'Pandas: Python data analysis library'. Accessed: 2025-02-14.
URL: *https://pandas.pydata.org/*

Naber, D. and andere (2025), 'Languagetool: A multilingual grammar and style checker'. Zugriff am 13. Februar 2025.
URL: *https://pypi.org/project/language-tool-python/*

Oliphant, T. and the NumPy Development Team (2025), 'Numpy: The fundamental package for numerical computation'. Accessed: 2025-02-14.
URL: *https://numpy.org/*

Ollama (n.d.*a*). [Online; accessed 2025-02-07].
URL: *https://ollama.com/search*

Ollama (n.d.*b*). [Online; accessed 2025-01-20].
URL: *https://ollama.com/search*

ollama/ollama-python: Ollama Python library (n.d.). [Online; accessed 2025-01-21].
URL: *https://github.com/ollama/ollama-python*

OpenSource.com (2024), 'What is open source?', https://opensource.com/resources/what-open-source. Accessed: 2024-12-02.

Overview - OpenAI API (n.d.*a*). [Online; accessed 2025-01-13].
URL: *https://platform.openai.com/docs/overview*

Overview - OpenAI API (n.d.*b*). [Online; accessed 2025-02-07].
URL: *https://platform.openai.com/docs/overview*

psutil · PyPI (2024). [Online; accessed 2025-01-21].
URL: *https://pypi.org/project/psutil/*

Research, G. (2025), 'Rouge score python library'. Zugriff am 13. Februar 2025.
URL: *https://pypi.org/project/rouge-score/*

Santhosh, S. (2023), 'Understanding bleu and rouge score for nlp evaluation | by sthanikam santhosh | medium'. [Online; accessed 2025-01-13].
URL: *https://medium.com/@sthanikamsanthosh1994/understanding-bleu-and-rouge-score-for-nlp-evaluation-1ab334ecadcb*

StudioLabs (2024), 'Open source for startups: Lower costs, higher growth'. Accessed: 2024-12-04.
URL: *https://www.studiolabs.com/open-source-for-startups-lower-costs-higher-growth/*

The Pros and Cons of Open-Source Software: A Guide for Developers and Executives (2023). [Online; accessed 2025-01-28].
URL: *https://www.bairesdev.com/blog/the-pros-and-cons-of-open-source-software-a-guide-for-developers-and-executives/*

to Wikimedia projects, C. (2001*a*), 'Css - wikipedia'. [Online; accessed 2025-01-23].
URL: *https://en.wikipedia.org/wiki/CSS*

to Wikimedia projects, C. (2001*b*), 'Javascript - wikipedia'. [Online; accessed 2025-01-23].
URL: *https://en.wikipedia.org/wiki/JavaScript*

to Wikimedia projects, C. (2006), 'Bleu - wikipedia'. [Online; accessed 2025-01-13].
URL: *https://en.wikipedia.org/wiki/BLEU*

Torvalds, L. (2024), 'Linus torvalds quotes', `https://www.brainyquote.com/quotes/linus_torvalds_135583`. Accessed: 2024-12-02.

Tran-Thien, V. (n.d.), 'Key criteria when selecting an llm'. [Online; accessed 2025-01-13].
URL: *https://blog.dataiku.com/key-criteria-when-selecting-an-llm*

W3Schools (2025), 'Typescript introduction'. Accessed: 2025-02-14.
URL: *https://www.w3schools.com/typescript/typescript¡ntro.php*

Waskom, M. and the Seaborn Development Team (2025), 'Seaborn: Statistical data visualization'. Accessed: 2025-02-14.
URL: *https://seaborn.pydata.org/*

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q. and Rush, A. M. (2020), 'Transformers: State-of-the-art natural

language processing', Online. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, 38–45.
URL: *https://www.aclweb.org/anthology/2020.emnlp-demos.6*