Mohammed V University
High National School for Computer Science
and Systems Analysis - ENSIAS RABAT

Software Engineering Class (GL)
End-of-Year Project Report

CodeGenPro

Participation in the programming of an AI-based plugin for interactive and automatic programming and compilation in Intellij, using the Chat-GPT API

Authors :
BELZEDI Ghizlane
DAOUDI Halima
Supervisor :
M. BAINA Karim

Jury:
M. BAINA Karim
Mme. BERRADA Bouchra

Academic Year: 2022-2023

To our dear parents, for their love and support, no dedication can be eloquent enough to express the love, esteem, dedication, and respect we have always had for them. Nothing in the world compares to the efforts they have made day and night for our education and well-being. This work is the result of their sacrifices made for our education and upbringing.

To our beloved brothers, sisters, and friends, words are not enough to express the love and affection I have for you. I dedicate this work to all the support and presence that have been of great help throughout my life.

To our dear teachers and mentors, we can never thank you enough for sharing your passion for teaching with us. We greatly appreciate your support, involvement, and experience throughout our academic year. We hope that this work lives up to your expectations.

To all those who have contributed directly or indirectly to the completion of this work.

Thank you.

- BELZEDI Ghizlane & DAOUDI Halima

Acknowledgments

As a token of appreciation, we would like to express our sincere thanks to everyone who contributed to the successful completion of our project and the development of this modest work. Our gratitude also goes to the members of the jury,**Ms. BERRADA Bouchra** and **Mr. BAINA Karim**, who agreed to evaluate this work and enrich it with their valuable remarks. We would also like to thank our supervisor for their guidance during the various phases of analysis, design, and implementation. Lastly, our thanks extend to all the professors and administrative staff of the National School of Computer Science and Systems Analysis (ENSIAS), as well as to all those who have contributed, directly or indirectly, to the realization of this work.

To everyone who has contributed, in any way, to the accomplishment of this project, we extend our heartfelt gratitude.

Abstract

Automated code generation is a valuable technique in software development that greatly reduces the time and effort required by developers to create new code. By automatically generating code based on given requirements, this approach proves to be highly efficient. Recently, ChatGPT, an advanced language model developed by OpenAI, has gained prominence for its ability to produce human-like responses across a wide range of text inputs, including those related to code generation. However, the extent of ChatGPT's effectiveness in generating code remains unclear, as its performance can be significantly influenced by the choice of prompt. To address these uncertainties, we conducted experiments to assess ChatGPT's capabilities for text-to-code generation.

Through the use of a chain-of-thought strategy and multistep optimizations, we devised prompts that guided ChatGPT's code generation process. Our findings demonstrated that by carefully designing prompts, we were able to greatly enhance the model's code generation performance. Furthermore, we analyzed the factors that impacted prompt design and provided valuable insights that can inform future research in this area.

Keywords : Large Language Models (LLM), Prompt Engineering, Code Assistant, Automatic Programming, Compilation, Chat-GPT API, Code Generation.

Résumé

La génération de code automatisée peut être une technique puissante pour le développement de logiciels, réduisant considérablement les efforts des développeurs et le temps nécessaire pour créer un nouveau code en le générant automatiquement en fonction des exigences. Récemment, le modèle de langage ChatGPT d'OpenAI est apparu comme un outil puissant pour générer des réponses semblables à celles d'un être humain à une large gamme d'entrées textuelles, y compris celles liées à la génération de code. Cependant,l'efficacité de ChatGPT pour la génération de code n'est pas bien comprise et les performances de génération pourraient être fortement influencées par le choix de la consigne. Pour répondre à ces questions, nous avons réalisé des expériences afin d'évaluer les capacités de ChatGPT en matière de génération de texte vers du code.

Nous avons conçu des consignes en exploitant la stratégie de la chaîne de pensée avec des optimisations en plusieurs étapes. Nos résultats ont montré qu'en concevant soigneusement des consignes pour guider ChatGPT, les performances de génération peuvent être considérablement améliorées. Nous avons également analysé les facteurs qui ont influencé la conception des consignes et fourni des connaissances qui pourraient orienter les recherches futures.

Mots clés : Grands modèles de langage (LLM), Ingénierie de prompts, Assistant de code, Programmation automatique, Compilation, API Chat-GPT, Génération de code.

Glossary

Acronyms	Meanings
AI	Artificial Intelligence
LLM	Large Language Model
API	Application Programming Interface
CA	Code Assistant
PE	Prompt Engineering
NLP	Natural Language Processing
T2C	Text-To-Code
NL	Natural Language
LM	Language Model
ACSE	Architecturally Compliant Software Engineering
AC	Answer Correct
APC	Answer Partly Correct
AIC	Answer InCorrect
EC	Explanation Correct
LM	Language Model
EPC	Explanation Partly Correct
EIC	Explanation InCorrect
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ASRs	Architecturally Signifi- cant Requirements
UML	Unified Modeling Language
SAAM	Software Architecture Analysis Method
GPT	Generative Pre-trained Transformer
PO	Product Owner
SM	Scrum Master
IDE	Integrated Development Environment
RLHF	Reinforcement Learning from Human Feedback

Table of contents

List of Figures

General Introduction

The ability to design instructions, known as prompts, has become essential for effectively communicating with large language models (LLMs) such as ChatGPT. The latter, developed by OpenAI, has revolutionized numerous fields through its ability to understand context and generate fluent and relevant responses. This document presents prompt engineering techniques that address common problems when interacting with LLMs. By utilizing prompt patterns, it is possible to customize outputs and interactions, thereby providing reusable solutions to enhance LLM performance in various tasks, from software engineering to code generation. These advancements open up new possibilities for harnessing the full potential of LLMs and adapting them to specific domains, ensuring precise and tailored results.

Chapter 1

Introduction to ChatGPT

1.1 Introduction

Large Language Models (LLMs) have demonstrated their value in various fields. ChatGPT, developed by OpenAI, has been trained using massive amounts of data and simulates human conversations by understanding context and generating appropriate responses. As a result, it has revolutionized many research and industrial domains. ChatGPT has shown great potential in software engineering to enhance various traditional tasks such as program repair, code understanding, and code generation. It has garnered significant attention due to its ability to effectively answer a wide range of human questions, with smooth and comprehensive responses surpassing previous public chatbots in terms of security and utility.

1.2 ChatGPT

ChatGPT is a language model (LM) developed by OpenAI and designed for conversational tasks such as question-answering and code generation. It is based on the GPT-3.5 series with 175 billion parameters and is optimized using reinforcement learning from human feedback. It can generate human-like responses based on the textual input from the user and an understanding of the context and conversation history. Furthermore, OpenAI is continuously improving ChatGPT by optimizing GPT-4.

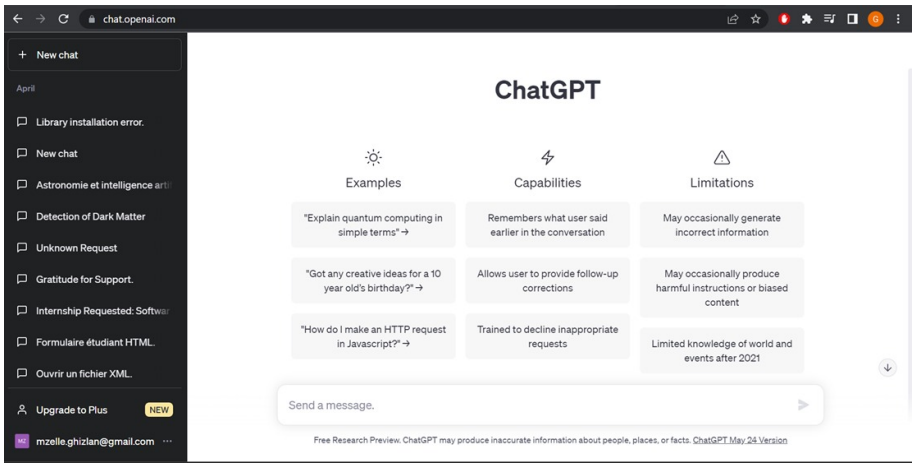

Figure 1.1: ChatGPT version of may 24th.

1.3 The Creation of ChatGPT

OpenAI LP, a research and deployment company with a limited goal, governed by the non-profit board of OpenAI, released ChatGPT on November 30, 2022. OpenAI is an artificial intelligence research lab founded by Elon Musk, Sam Altman, Greg Brockman, Ilya Sutskever, Wojciech Zaremba, and John Schulman. This company is supported by investors, including Microsoft, the charitable foundation of Reid Hoffman, and Khosla Ventures.

1.4 The Operation of ChatGPT

Once the user's input is received by ChatGPT, machine learning algorithms are applied to a large amount of text to generate a response similar to that of a human. ChatGPT operates on a neural network-based architecture called Transformer.

Datasets from websites, books, and articles are used to train the model on language patterns and structure. It learns to predict the next word based on the preceding word.

Once trained, the model can generate new text by predicting the next word in a sentence, given a prompt or context. The process is repeated until the model has generated a complete sentence or the required number of words. The model also utilizes an attention mechanism during text generation. This mechanism allows it to selectively focus on certain parts of the input for more accurate and coherent responses.

When using ChatGPT for conversational intelligence, the model is typically fine-tuned on a smaller dataset of conversational text to further enhance its ability to generate human-like responses.

Figure 1.2: The Operation of ChatGPT.

- **Input Processing:**
  Let's take the example where we ask ChatGPT the question "What is the capital of Morocco?" ChatGPT starts by tokenizing the input into a sequence of words: "What", "is", "the", "capital", "of", "Morocco", "?". Then, it converts each token into a vector representation using word embeddings. These embeddings capture the meaning of each word in the context of the sentence.

- **Contextual Analysis:**
  Next, ChatGPT analyzes the sequence of word vectors and their mutual relationships to understand the meaning of the input in the context of the conversation. In this case, it recognizes that the user is asking a question about the capital of a country.

- **Response Generation:**
  Based on its understanding of the input, ChatGPT generates a response by predicting the most probable sequence of words that would follow the input in a natural conversation. In this case, it could generate the following response: "The capital of Morocco is Rabat."

- **Output Processing:**
  Finally, ChatGPT converts the sequence of word vectors representing the response into natural language text and presents it to the user: "The capital of Morocco is Rabat."

1.5 How is ChatGPT trained?

Abundant amounts of conversational exchanges, tailored to a specific task or domain, are utilized as a dataset to train ChatGPT. A variant of the transformer architecture is employed to predict words. Instead of providing explicit responses to the model, it is allowed to independently discern patterns and relationships within the data. ChatGPT is the outcome of fine-tuning GPT-3.5 through supervised learning and reinforcement techniques.

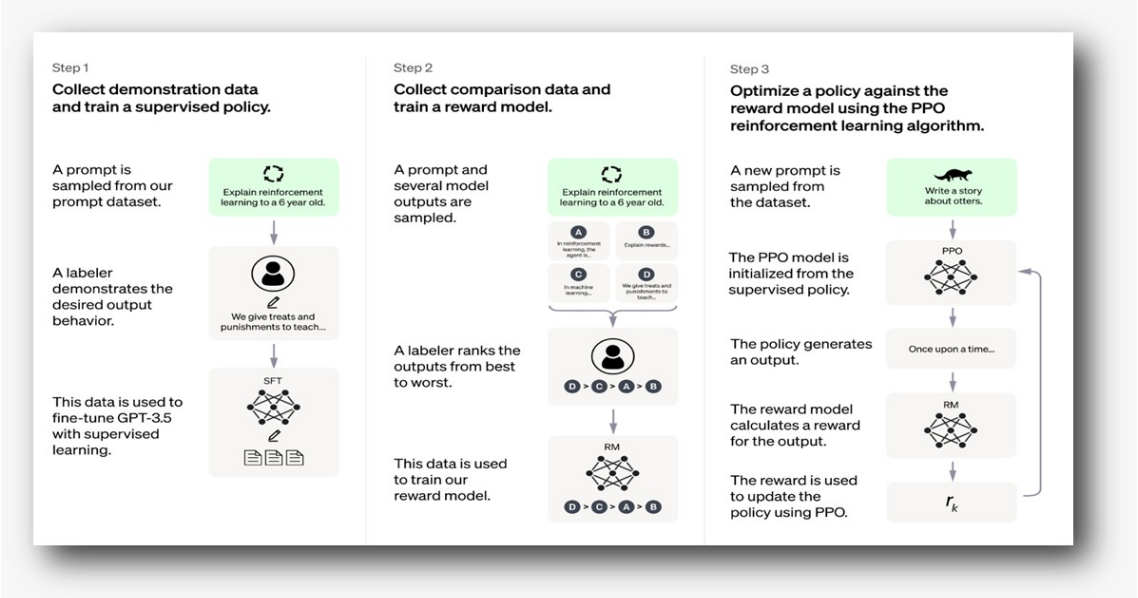

Figure 1.3: Training Method - ChatGPT.

ChatGPT, just like its associated model InstructGPT, is trained using reinforcement learning from human feedback (RLHF). There is only a slight difference in the data collection setup for ChatGPT. The model training involved supervised fine-tuning, where AI trainers played the roles of both the user and the AI assistant in the conversation. Trainers composed their responses based on suggestions provided by the model. The new dialogue dataset was then combined with the InstructGPT dataset and transformed into a dialogue format. Creating a reward model for reinforcement learning required comparison data, consisting of model responses ranked by quality.

A selection of conversations between the AI trainers and chatbots was used to collect this data. A randomly selected message produced by the model was chosen, and multiple completions of the message were sampled for the trainers to rank. Based on these reward models, ChatGPT was fine-tuned using proximal policy optimization. Multiple iterations of this process were performed.

1.6 ChatGPT API

The ChatGPT API is a programming interface that provides access to the power and functionality of the state-of-the-art language model developed by OpenAI. This API offers developers the opportunity to integrate ChatGPT into their own applications, products, or services, enabling them to leverage the advanced text generation capabilities of ChatGPT. Whether it's creating interactive chatbots, conversational agents, or other natural language processing-based applications, the ChatGPT API provides a powerful and flexible solution to enhance the user experience and deliver more natural and seamless interactions. In this introduction, we will explore the features and benefits of the ChatGPT API, as well as the various ways it can be used to meet the specific needs of developers and users.

1.7 The Functioning of the ChatGPT API

The ChatGPT API simplifies and expedites the creation of chatbots by providing a quick and easy solution. Leveraging the advanced capabilities of the ChatGPT model, developers can design chatbots that appear natural and can handle complex queries.

Using this API, chatbots can process real-time text or voice messages, and they are even capable of handling intricate requests. Furthermore, developers have the option to create sophisticated chatbot scripts that can provide consistent responses across multiple sessions. With the ChatGPT API, developers have the power to create intelligent and user-friendly chatbot applications that cater to a variety of needs.

1.8 Advantages of the ChatGPT API

The ChatGPT API offers various advantages for developing an intelligent chat platform based on AI algorithms:

With the ChatGPT API, it is possible to create natural and engaging chatbot dialogues and even train custom chatbot models.

The API is powerful and provides features such as automatic response generation, conversation management, dialogue dynamics, dialogue flow, and much more.

This not only allows for the creation of an intelligent chatbot platform but also opens up many exciting opportunities for users. It is even possible to develop a customizable chatbot system tailored to specific user needs.

The ChatGPT API enables seamless integration of the chatbot into existing organizational systems and applications.

With the ChatGPT API, developers can enhance the conversational capabilities of their applications and deliver a more natural user experience.

1.9 Conclusion

In conclusion, our exploration of ChatGPT has provided us with a comprehensive understanding of this AI-based conversational text generation system developed by OpenAI. ChatGPT operates through Reinforcement Learning from Human Feedback (RLHF) and undergoes supervised fine-tuning with human AI trainers. This training process involves data collection and the creation of reward models for reinforcement learning. With its promising potential, ChatGPT pushes the boundaries of conversational text generation. Additionally, the ChatGPT API emerges as a powerful tool for creating intelligent chatbots, offering various functions and user-friendly features. It enables the simulation of human-like conversations, making it an appealing option for chatbot development. However, it is important to stay informed about alternative options and keep an eye on the availability of the ChatGPT API to ensure the best choices for chatbot creation.

Chapter 2

Failure Analysis & Design Solutions

2.1 Introduction

Large Language Models (LLMs), such as ChatGPT, have demonstrated remarkable proficiency in various natural language processing (NLP) tasks, including automatic translation, question answering, text summarization, natural language understanding, and code generation.

However, a comprehensive analysis of ChatGPT's failures is lacking, which is the objective of this study. We present categories of failures, including programming errors as well as limitations of ChatGPT.

To further improve the performance of Large Language Models (LLMs) in code generation evaluation, we conducted an investigation into several solicitation methods. Our results indicate that using this latest method, LLMs like ChatGPT can generate code with fewer errors and higher quality. Our findings aim to provide a preliminary experience for appropriately assessing code generation on ChatGPT while offering a variety of techniques (Prompt Engineering) for instruction design to enhance ChatGPT's accuracy and reliability.

2.2 Statistics.

In this section, we present a case study examining the characteristics of incorrect responses in terms of code generation, how ChatGPT's responses can change when we provide it with more information, and an example of an inconsistent response-explanation pair. The goal of our case study is to gather insights into the reasons why ChatGPT makes mistakes and explore how certain solicitation strategies can lead to a higher probability of correct responses.

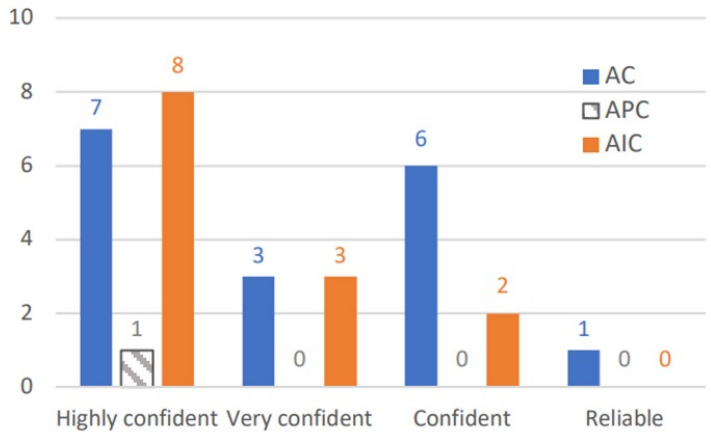

Figure 2.1: Confidence Levels Reported to ChatGPT for Correct, Partially Correct, and Incorrect Responses in Terms of Code Generation.

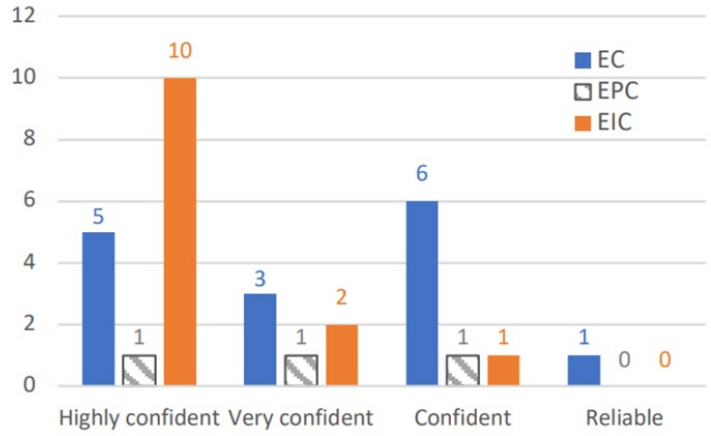

Figure 2.2: Confidence Levels Reported by ChatGPT for Correct, Partially Correct, and Incorrect Explanations.

Based on statistics from ArXiv, an online archive of scientific preprints, we performed an experiment using one randomly selected incorrect response out of a total of 14. Subsequently, we modified the prompt with the intention of prompting ChatGPT to generate the correct response. The previous statistics led us to conclude and identify three main reasons for ChatGPT's incorrect responses:

- **Lack of Knowledge in ChatGPT:** The first category involves situations where ChatGPT may lack the necessary knowledge to solve the problems posed to it. ChatGPT appeared to lack definitions for concepts such as "defect," "failure," and "error," which led to incorrect treatment of errors as failures or malfunctions.
- **Making a Wrong Assumption:** ChatGPT can also focus on an irrelevant part of the question and provide an incorrect answer because it makes an incorrect assumption about what is important.
- **Both Factors Combined:** ChatGPT seems to both lack knowledge and make incorrect assumptions simultaneously. For example, ChatGPT may make an erroneous assumption about a defect in a program while lacking knowledge about what an error is. As a result, ChatGPT's response regarding the error in that program would be incorrect due to these two characteristics.

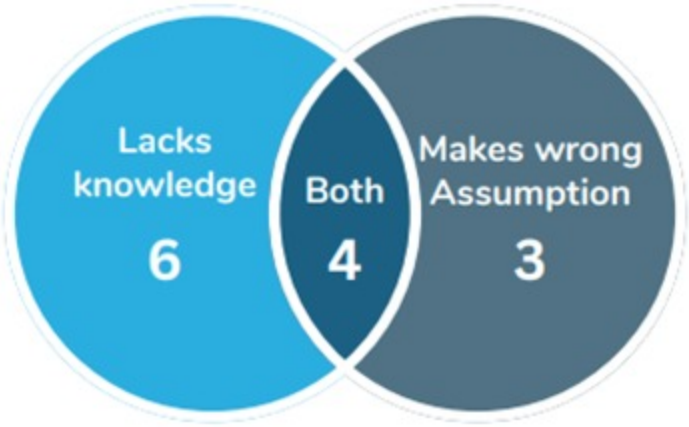

Figure 2.3: Venn Diagram of Characteristics of Incorrect Responses by ChatGPT.

The observation that ChatGPT can focus on an irrelevant part of the question and generate incorrect code has led us to consider a solution to address this comprehension problem. This observation has motivated us to work on this project with the goal of finding ways to improve the relevance of code generation provided by ChatGPT.

2.3 Problem Statement

The effectiveness of ChatGPT for code generation is not well understood, and the generation performance can be highly influenced by the choice of prompts. Hence, there is a need to automate code generation in a way that makes it a powerful technique for software development, significantly reducing the efforts and time required by developers to create new code by generating it automatically based on the requirements.

2.4 Project Objectives

2.4.1 General Objectives

The primary goal of this report is to provide developers with guidance on leveraging large language models (LLMs) within their software applications. We aim to achieve this by introducing a website or tool that enables interactive and automated code generation using prompts to develop LLM responses. Additionally, we will share best practices for effectively formulating prompts and explore common use cases, including code generation. By the end of this report, developers will have a comprehensive understanding of utilizing LLMs through API calls and practical insights for enhancing their software development process.

- Enable developers to leverage the capabilities of large language models (LLMs) effectively and efficiently within their software applications.
- Improve the user experience by providing an intuitive interface that enhances the understanding and integration of user requirements.
- Foster collaboration between designers and the LLM through an interactive agent, promoting seamless integration of designer requirements into the coding process

- Streamline the code generation process by automating repetitive tasks, saving time, and reducing the potential for errors.
- Establish best practices and guidelines for prompt formulation, ensuring consistent and high-quality interactions with the LLM.
- Encourage the adoption of LLMs in software development by demonstrating their value in code generation and other common use cases.

2.4.2 Specific Objectives

- Develop an API interface that enables developers to make seamless calls to a large language model (LLM) within their software applications.
- Implement a Software Blueprint integration to provide predefined patterns and templates for code generation, enhancing the efficiency and consistency of development.
- Enhance the prompt formulation process by incorporating best practices and guidelines, ensuring effective communication of requirements to the LLM.
- Improve the relevance and quality of prompts used when interacting with the LLM, considering context, specificity, and clarity to generate accurate and desired responses.
- Build an interactive agent utilizing ChatGPT, allowing designers to collaboratively code with the LLM iteratively, enabling efficient communication and integration of their requirements.
- Automate the code generation process using the LLM, reducing manual effort and increasing development speed and productivity.

2.5 Context

Language models like ChatGPT have been trained on a vast corpus of online text, which means they possess extensive knowledge on many topics, but they may also have gaps or biases based on the distribution of languages "ingested" by the language model and the associated representations analyzed by AI. This project targets developers interested in creating projects without coding. It will focus on prompts tailored to instructions, which are easier to use and become safer and more aligned, as opposed to basic language models.

2.6 Approach and Methodology

To achieve this objective, it is important to understand the limitations and strengths of the language model. The prompt needs to be clear, precise, and well-structured to avoid misunderstandings and incorrect responses. The presentation of the prompt input can also influence the quality of the response, so it is important to use appropriate capitalization and punctuation.
It is also recommended to provide context to the prompt input. The more context provided, the more precise and relevant the response can be. For example, if you want to

obtain information on a specific topic, you can include additional information in your prompt to help ChatGPT understand what you are looking for. Lastly, it is important to avoid sensitive topics or prompts that could be deemed offensive, political, or violent, as these can push the boundaries of the learning ChatGPT has undergone. It is always best to exercise caution when using language models.
To further improve the quality of the generated content, prompts can also be tailored to specific writing styles or authors, and the context of the request should be described in detail to ensure the best possible response.
The report will cover best practices for prompt usage in software development and common use cases for language models. The tools have been carefully thought out, validated, and selected. The approach will involve teaching developers how to provide clear instructions to a language model, similar to giving instructions to an intelligent person who is unfamiliar with task specifics. The project aims to stimulate developers' imagination regarding new applications that can be built using language models.

2.7 Gantt Chart

Here is the Gantt chart of our project.

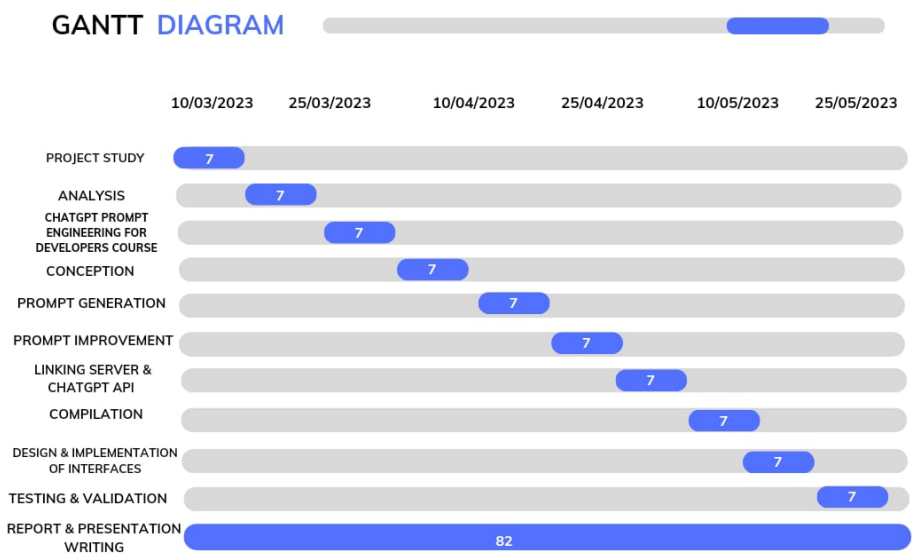

Figure 2.4: Gantt Chart

2.8 Use Case Diagram

The use case diagram is a diagram that highlights the functional relationships between actors and our studied system. As we have just seen, we have identified three actors who interact with the system represented in the use case diagram below.

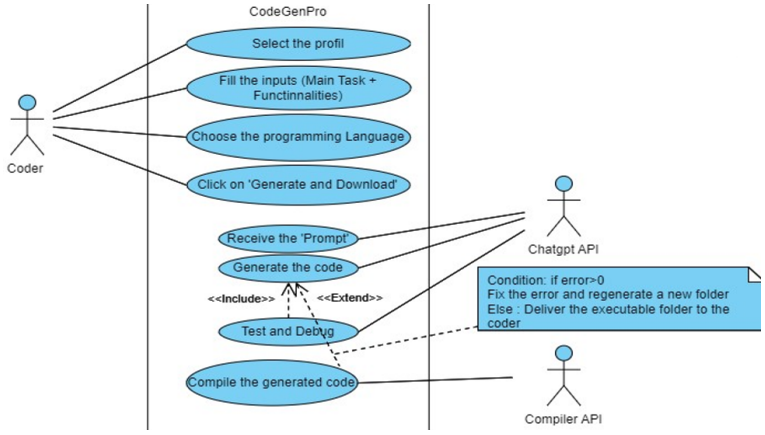

Figure 2.5: Use Case Diagram.

2.9 Sequence Diagram

A sequence diagram is a visual representation that shows the interactions and order of messages exchanged between objects or actors in a system. It illustrates the dynamic behavior, timing, and dependencies of these interactions, helping to depict scenarios in a clear and concise manner.

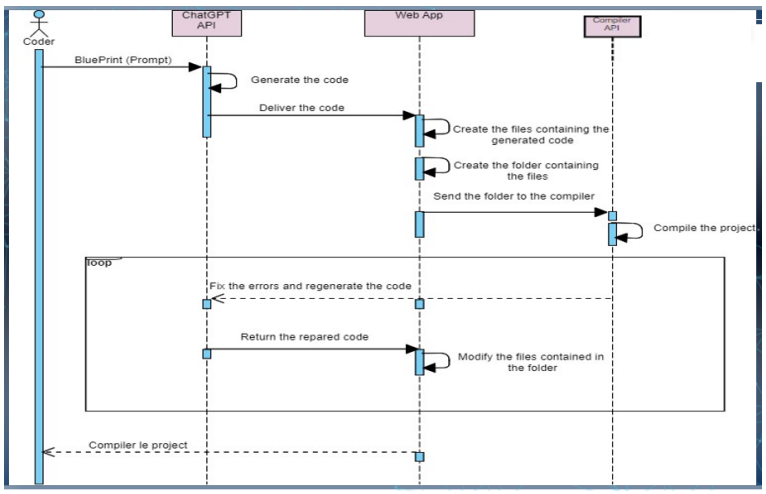

Figure 2.6: Sequence Diagram.

2.10 Conclusion

In conclusion, this report aims to provide a graphical interface for developers to generate code-free projects by injecting a prompt into a blueprint. It provides essential information on the use of large language models (LLMs) in their software applications. It highlights the importance of understanding the limitations and strengths of the language model to formulate clear and precise prompts, avoiding incorrect responses. The integration of an improved and relevant prompt, as well as providing additional context, is recommended to

achieve more accurate and relevant responses. The report will also address best practices for prompt usage in software development, while encouraging developers to imagine new applications based on language models. The adopted approach is to guide developers in formulating clear instructions for language models, aiming to optimize their usage and facilitate high-quality results.

Chapter 3

Prompt Engineering and Collaborative Architecting

3.1 Introduction

Prompt engineering is an increasingly important skill for effectively interacting with large language models (LLMs) like ChatGPT. Prompts are instructions given to an LLM to enforce rules, automate processes, and ensure specific qualities (and quantities) of generated output. Prompts are also a form of programming that allows customization of results and interactions with an LLM. This chapter describes a catalog of prompt engineering techniques presented as patterns that have been applied to address common problems in LLM interactions. Prompt patterns are a knowledge transfer method similar to software patterns, as they provide reusable solutions to common problems encountered in a specific context, namely output generation and interaction when working with LLMs. This chapter contributes the following to research on prompt engineering applied to software development automation tasks using LLMs. It explains how to construct prompts using multiple patterns and illustrates beneficial combinations of prompt patterns

3.2 Architecting in Harmony: Collaborative Human-Bot Approach

Software Architecture, as defined in the ISO/IEC/IEEE 42010:2011 standard, aims to simplify complexities associated with source code-based implementations by utilizing architectural components and connectors. These elements serve as a blueprint for the development of software applications, services, and systems. Architecture-centric approaches, which leverage architectural knowledge such as patterns, styles, languages, and frameworks, have proven to be valuable in both academic and industrial projects, enabling efficient and effective software design and development.
To support software designers and architects in systematically and incrementally designing software architectures, there is a need for an architecting process, also known as the process for architecting software. This process enables the systematic and incremental design of software architectures, providing a framework for designers and architects to follow.

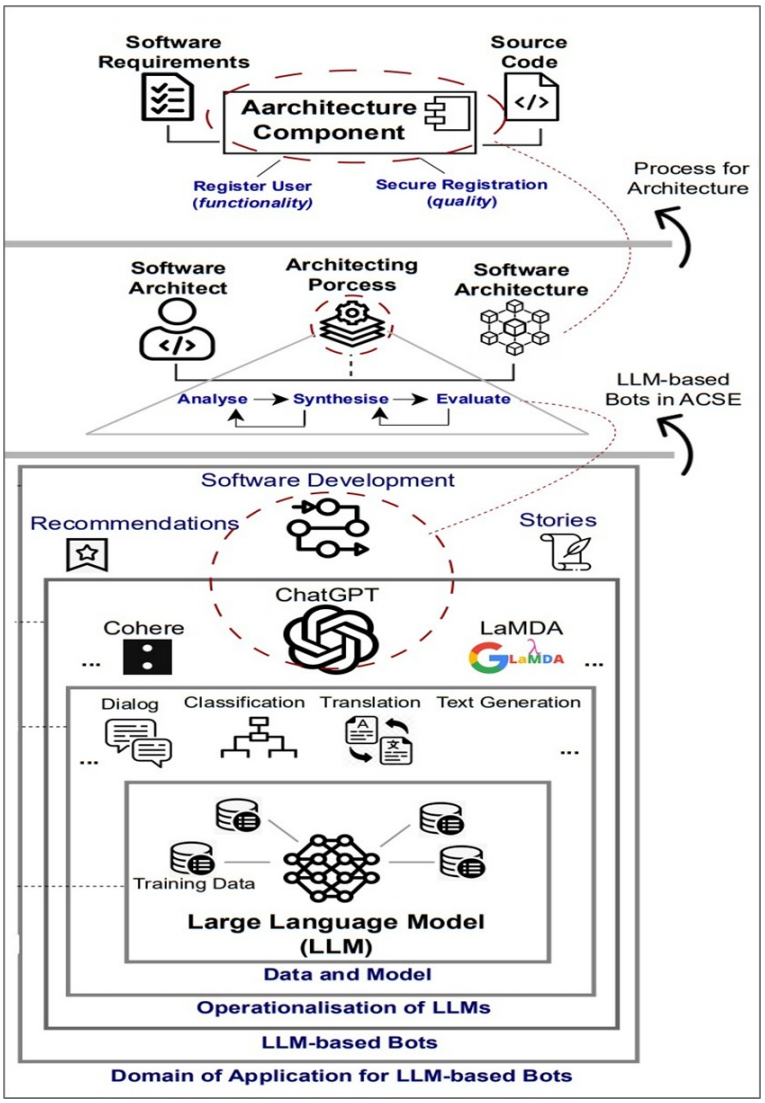

Figure 3.1: Collaborative Architecting with Human-Bot Integration.

1. Software architecture Designing a blueprint for software development that involves mapping requirements (functionality, quality, constraints, etc.) to source code implementations as architectural elements (components). This blueprint encompasses architecturally significant requirements such as functionality, quality, and constraints, and guides the process of translating them into tangible components of the software system.

2. Software Architecting Process Facilitating designers and developers in adopting a structured and methodical approach through various architectural activities to enable ACSE (Architecturally Compliant Software Engineering). These activities include analysis (requirements, constraints, etc.), synthesis (modeling, specification, etc.), and evaluation (validation, testing, etc.). By following this approach, designers and developers are empowered to effectively and systematically address the necessary steps for ACSE.

3. Application Scope for LLM-based Bots

    - **Domain of application :** Scenarios where LLM-based bots can be applied include recommender systems, storytelling, and automating as well as facilitating software development.

- **Bots built on LLMs:** Agents capable of engaging with humans to offer intelligence and decision-making assistance to human users.
- **Use cases of LLMs:** Collaborative Human-Bot Dialogue, Content Categorization, Language Translation, and Text Generation with Extensive Data Analytics and Processing.
- **Large Language Models (LLMs):** Utilizing deep learning techniques and algorithms to identify, condense, and generate content by leveraging knowledge acquired from extensive datasets.

3.3 Exploring the Methodological Landscape: Approaches to Research

In this section, we outline the comprehensive research methodology, consisting of three primary stages.

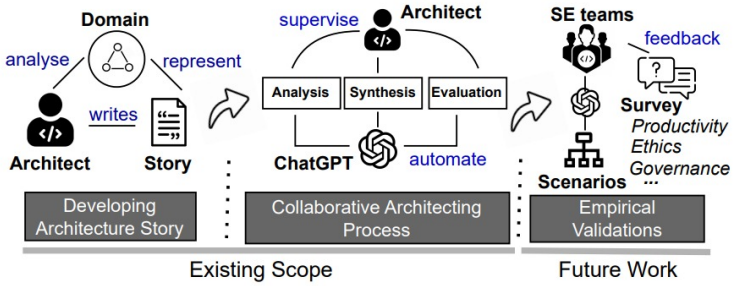

Figure 3.2: An Insight into the Methodology Employed in the Research.

- **Phase 1** - Architecture Story Development: In this phase, a software architecture story is created by analyzing the software domain and expressing the core functionality, desired quality, and constraints in natural language. This story serves as a foundation for the architecting process and is used as a prompt for collaborative architecting with ChatGPT.
- **Phase 2**- "Enabling Collaborative Architecting," it consists of three activities.
  - Architectural analysis involves ChatGPT articulating architecturally significant requirements (ASRs) through automatic generation, manual specification, or iterative refinement.
  - Architectural synthesis utilizes UML to create an architecture model, incorporating reuse knowledge, best practices, tactics, and patterns.
  - Architectural evaluation assesses the synthesized architecture against ASRs using scenarios from the architectural story, employing SAAM to guide ChatGPT's evaluation process .
- **Phase 3** - "Empirical Validations," extends the study by conducting empirical validations of collaborative architecting. It explores the role of ChatGPT in human-bot collaborative software architecting and outlines future work on addressing sociotechnical challenges through empirically grounded guidelines.

3.4 What is a blueprint ?

In the context of prompt patterns, a blueprint refers to a structured framework or guide that outlines reusable solutions to common challenges encountered when interacting with large language models (LLMs) like ChatGPT. It provides a structured way of discussing and implementing prompting solutions, offering a roadmap for effectively and efficiently utilizing LLMs in various tasks and domains. The blueprint captures the essence of prompt patterns, allowing users to follow a systematic approach in designing prompts for optimal interactions with LLMs.

3.5 Blueprint Structure for Designing Prompt Patterns

1. Problem Statement:
    - Challenge: Generating coherent and contextually relevant text (Code) responses from a language model.
    - Problem: Lack of control over response quality and coherence in open-ended conversations.
2. Solution Description:
    - Approach: Utilize a system message prompt pattern.
    - Solution: Incorporate a system message before the user's input to guide the language model's response.
    - Design: Begin the prompt with a system message that sets the desired context or behavior for the conversation.
3. Implementation Steps:
    - Step 1: Define the desired context or behavior to be conveyed in the system message.
    - Step 2: Place the system message before the user's input when making an API call to the language model.
    - Step 3: Tailor the system message based on the specific task or conversation requirements.
4. Best Practices and Guidelines:
    - Keep the system message concise, clear, and relevant to the desired conversation context.
    - Experiment with different system messages to find the most effective prompts for your specific use case.
5. Evaluation and Validation:
    - Measure the quality and coherence of responses generated using the prompt pattern through manual evaluation or user feedback.
    - Fine-tune and iterate the system message prompt pattern based on the evaluation results to improve response performance.

3.6 What is a prompt ?

A prompt is a set of instructions that are given to a language model (LLM) in order to customize, enhance, or refine its capabilities. By providing specific rules and guidelines, a prompt can influence the subsequent interactions and output generated by the LLM. It sets the context for the conversation and informs the LLM about what information is important and how the desired output should be in terms of form and content

3.7 ChatGPT and Prompt Engineering

With the emergence of language models like ChatGPT with a large number of parameters (over 100 million), their advanced ability to generate text has led to the development of a new approach called prompt engineering (PE) in the field of natural language processing (NLP). The goal of PE is to design an appropriate prompt for a pre-trained model and achieve highperformance predictions.

In PE, a prompt $x' = f_{prompt}(x) \in x$ is created for a given textual input x (e.g., "write a Java code for converting int to string") that describes a specific task (e.g., code generation). Using this prompt x, the language model performs the prediction $y = f_{LLM}(x') \in y$. There are two primary tasks in prompt engineering:

1. Prompt Template Engineering: This task involves designing an appropriate template $x'$for the language model input (e.g., "write a Java code for $[x]$," where "$[x]$" represents a variable for natural language description). The performance of the language model's prediction y is sensitive to the sentence(s) designed in the template.

2. Prompt Answer Engineering: The objective of this task is to design an answer space Z within the prompt, allowing for the generation of better answers y from a limited scope $y \in Z$ (e.g., "Which code is better ? A or B ?"). Advanced prompt engineering tasks involve manipulating multi-prompt scenarios, such as prompt augmentation and composition.

The prompt (x') can be generated in four ways (f_{prompt}):

1. Manual Construction: This approach is suitable for template-based prompts and situations where the prompt is uncomplicated or requires few-shot prompting.

2. LM Generation: Leveraging the language model, prompts (x') are generated to match each textual input (x).This method compensates for the limitations of manual construction.

3. Retrieval-Based Prompt: This method relies on well-annotated external resources (e.g., Wikipedia) to address the instability issue associated with prompt generation.

4. Prompt Learning: A supervised model is built to automatically update the prompt based on the language model's generation and the associated ground truth.

3.8 Code Generation Tasks

Code generation involves the automatic creation of code based on a given requirement specification. This process can be highly beneficial for developers as it automates repetitive programming tasks, saving them time and effort. The specification for code generation

can be expressed in various formats. In this particular study, we focused on investigating a specific type of code generation task known as Text-to-Code (T2C) Generation. This task involves taking a functional specification written in natural language (NL), such as a textual description, and using a code generation model to produce code (e.g., Java) based on that description.

3.9 Prompt Pattern Catalog

Pattern Category	Prompt Pattern
Input Semantics	*Meta Language Creation*
Output Customization	*Output Automater* *Persona* *Visualization Generator* *Recipe* *Template*
Error Identification	*Fact Check List* *Reflection*
Prompt Improvement	*Question Refinement* *Alternative Approaches* *Cognitive Verifier* *Refusal Breaker*
Interaction	*Flipped Interaction* *Game Play* *Infinite Generation*
Context Control	*Context Manager*

As depicted in the table, our classification framework identifies five categories of prompt patterns: Input Semantics, Output Customization, Error Identification, Prompt Improvement, and Interaction. Each category is described below.

1. The **Input Semantics** category focuses on how an LLM understands and translates the input into usable information for generating output. It includes the Meta Language Creation pattern, which involves creating a custom language that better suits the user's ideas when communicating with the LLM.

2. The **Output Customization** category revolves around constraining or tailoring the characteristics of the generated output. Prompt patterns in this category include Output Automater, Persona, Visualization Generator, Recipe, and Template.

    - **Output Automater** allows users to create scripts for automating tasks suggested by the LLM.
    - **Persona** assigns a specific role or persona to the LLM during output generation.
    - **Visualization Generator** enables the generation of visualizations through textual outputs that can be used with other tools.
    - **Recipe** provides a sequence of steps or actions to achieve a desired result.

- **Template** allows users to specify an output template for the LLM to fill in with content.

3. **The Fact Check List**prompts the LLM to generate a list of facts on which the output depends, which can be fact-checked.
    - **The Fact Check List**prompts the LLM to generate a list of facts on which the output depends, which can be fact-checked.
    - **The Reflection pattern**requires the LLM to introspect on its output and identify any errors.
4. The **Prompt Improvement** category aims to enhance the quality of both input and output. It includes the Question Refinement, Alternative Approaches, Cognitive Verifier, and Refusal Breaker patterns.
    - **Question Refinement**ensures that the LLM always suggests an improved version of the user's question.
    - **Alternative Approaches**prompts the LLM to suggest alternative methods for achieving a given task.
    - **Cognitive Verifier** instructs the LLM to propose subquestions for the user to answer, which are then combined to provide an answer to the overall question.
    - **Refusal Breaker**involves the LLM rephrasing the user's question if it refuses to provide an answer.
5. **The Interaction category** focuses on the interaction between the user and the LLM. It includes the Flipped Interaction, Game Play, and Infinite Generation patterns.
    - **Flipped Interaction**prompts the LLM to ask questions instead of generating output.
    - **Game Play**involves the LLM generating output in the form of a game.
    - **Infinite Generation**instructs the LLM to generate output indefinitely without requiring the user to reenter the generator prompt each time.
6. the **Context Control** category focuses on controlling the contextual information within which the LLM operates. It includes the Context Manager pattern, which allows users to specify the context for the LLM's output.

3.10 Contextual statement of prompt pattern

3.10.1 The Meta Language Creation Pattern

This pattern helps users create prompts using an alternate language for better communication with language models. It explains the semantics of the new language to the user.

- Structure and Key Ideas :

Contextual Statements
When I say X, I mean Y (or would like you to do Y)

- Consequences:
  Although the Meta Language Creation pattern empowers users to customize their interaction with an LLM, it can introduce confusion. Clear definition of language semantics is essential while avoiding ambiguities that hinder the LLM's performance. Certain symbols or terms may create confusion and prompt warnings from the LLM. While this pattern enhances accuracy and utility, caution must be exercised, especially when describing intricate concepts. It is advisable to use a single metalanguage per conversation session to prevent conflicting or unexpected semantics from arising over time.

3.10.2 The Output Automater Pattern

The Flipped Interaction pattern shifts the conversation dynamic, empowering the LLM to ask questions and guide the discussion towards a specific objective. Instead of the user directing the conversation, this approach allows the LLM to utilize its knowledge to extract information more precisely.

- Structure and Key Ideas :

Contextual Statements
I would like you to ask me questions to achieve X
You should ask questions until this condition is met or to achieve this goal (alternatively, forever)
(Optional) ask me the questions one at a time, two at a time, etc.

- Consequences:
  Although the Meta Language Creation pattern empowers users to customize their interaction with an LLM, it can introduce confusion. Clear definition of language semantics is essential while avoiding ambiguities that hinder the LLM's performance. Certain symbols or terms may create confusion and prompt warnings from the LLM. While this pattern enhances accuracy and utility, caution must be exercised, especially when describing intricate concepts. It is advisable to use a single metalanguage per conversation session to prevent conflicting or unexpected semantics from arising over time.

3.10.3 The Persona Pattern

This pattern enables users to specify a desired perspective or point of view for the LLM's output. By assigning a persona, such as a security expert, users can guide the LLM to generate output focused on relevant aspects, such as during code review.

- Structure and Key Ideas :

Contextual Statements
Act as persona X
Provide outputs that persona X would create

- Consequences:
  An intriguing aspect of adopting non-human personas is that the LLM can make fascinating assumptions or "hallucinations" about the context. One popular example

involves instructing ChatGPT to simulate a Linux terminal and generate the expected output for various commands, including listing files and running operations on them.

3.10.4 The Question Refinement Pattern

This pattern actively involves the LLM in the process of crafting prompts, aiming to provide users with improved and refined questions instead of their original ones.

- Structure and Key Ideas :

Contextual Statements
Within scope X, suggest a better version of the question to use instead
(Optional) prompt me if I would like to use the better version instead

- Consequences:
  The Question Refinement pattern improves interactions by aligning user knowledge with the LLM's understanding, but it may overly narrow down questions and overlook important information.

]The Alternative Approaches Pattern

The intent of the Alternative Approaches pattern is to ensure that the LLM always suggests alternative methods for completing a task, prompting users to explore beyond their familiar approaches. This pattern helps users overcome cognitive biases and discover better ways to solve problems by introducing them to alternative approaches they may be unaware of.

- Structure and Key Ideas :

Contextual Statements
Within scope X, if there are alternative ways to accomplish the same thing, list the best alternate approaches
(Optional) compare/contrast the pros and cons of each approach
(Optional) include the original way that I asked
(Optional) prompt me for which approach I would like to use

- Consequences:
  An intriguing aspect of adopting non-human personas is that the LLM can make fascinating assumptions or "hallucinations" about the context. One popular example involves instructing ChatGPT to simulate a Linux terminal and generate the expected output for various commands, including listing files and running operations on them.

3.10.5 The Cognitive Verifier Pattern

The Cognitive Verifier pattern is designed to enhance LLMs' reasoning capabilities by breaking down questions into smaller, individual questions that collectively yield improved answers.

- Structure and Key Ideas :

Contextual Statements
When you are asked a question, follow these rules
Generate a number of additional questions that would help more accurately answer the question
Combine the answers to the individual questions to produce the final answer to the overall question

- Consequences:
  The number of questions generated can be determined explicitly or left to the discretion of the LLM. Specifying an exact number can restrict the amount of additional information required from the user, ensuring it remains within their willingness and capacity to contribute. However, this approach may overlook potentially valuable questions that fall outside the predetermined count.

3.11 Conclusion

Prompt patterns serve as reusable solutions to common user challenges when interacting with LLMs for various tasks. Lessons learned from the work highlight the enrichment of LLM capabilities through prompt patterns, the need for ongoing refinement and expansion of the pattern catalog, the evolution of LLM capabilities requiring pattern adjustments, and the generalizability of prompt patterns across different domains. The paper encourages further research and development to enhance prompt pattern design and unlock new capabilities in conversational LLMs.

Chapter 4

Project Realization

4.1 Introduction

In this chapter, we present the implementation and execution part of our project. We will begin by introducing the work environment and the development tools used. Then, we will provide an overview of the various graphical interfaces.

4.2 Work Methodology

4.2.1 Agile Scrum

As part of the development of this project, we have chosen to implement the SCRUM agile management methodology. Tasks are estimated in terms of time and complexity. These estimations help in both planning deliveries and estimating the cost. The features, also known as "user stories," that are the focus of a sprint, make up the "sprint backlog" of the potentially deliverable product at the end of the sprint.

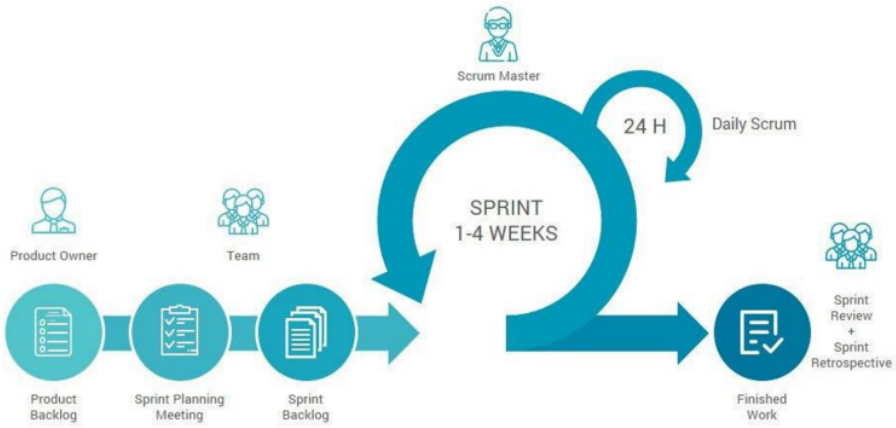

Figure 4.1: The Scrum Methodology

4.2.2 Roles in SCRUM:

In SCRUM, there are three roles: the Product Owner (PO), the Scrum Master (SM), and the Coder .

The Product Owner acts as the developer persona and expresses the software blueprint to the Scrum Master. The Scrum Master then encapsulates the blueprint into a well-crafted prompt and sends it to the Coder. The Coder generates the code based on the provided prompt. The Scrum Master compiles the code using the compiler specified by the Product Owner and provided in the Integrated Development Environment (IDE). This compilation process is performed to verify if the code successfully compiles and aligns with the blueprint, which is tested through a functional test scenario provided by the Product Owner.

If the requirements are met, the Scrum Master sends the code back to the Product Owner. However, if there are issues with compilation or non-compliance with the functional test, the Scrum Master returns to the Coder to address the compilation problem or the non-conformity issue based on the test scenario.

4.2.3 GitHub

GitHub is a platform for collaborative software development that relies on the Git version control system. It offers developers an online workspace where they can store, manage, and share their source code.

Through GitHub, developers have the ability to create repositories specifically for their projects. They can track and monitor changes made to the code, propose modifications through pull requests, and engage in efficient collaboration. GitHub's robust tools and vibrant community have established it as a vital platform for collaborative and transparent software development.

Figure 4.2: GitHub

4.2.4 Tools Used

HTML (HyperText Markup Language)

Figure 4.3: HTML

HTML (HyperText Markup Language), commonly abbreviated as HTML or in its latest version HTML5, is the markup language designed for displaying web pages.

CSS (Cascading Style Sheets)

Figure 4.4: CSS

CSS (Cascading Style Sheets) is used to describe the presentation of HTML and XML documents. The standards defining CSS are published by the World Wide Web Consortium (W3C).

JS (JavaScript)

Figure 4.5: JS

JavaScript is a scripting or programming language that allows you to implement complex functionality on web pages. Whenever a web page goes beyond displaying static information, such as periodic content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc., JavaScript comes into play. It is the third layer of the standard web technology stack, with two other layers (HTML and CSS) covered in much more detail in other parts of the learning space.

React JS

Figure 4.6: React JS

React JS is a popular JavaScript library for building user interfaces. It provides a component-based architecture, allowing developers to create reusable UI components. React JS is widely used for building interactive and dynamic web applications. It simplifies the process of managing the application state and efficiently updating the user interface based on changes in the data. React JS is an integral part of modern web development and is often used in combination with other tools and libraries to create powerful and scalable applications.

OpenAI

Figure 4.7: OpenAI

OpenAI is a leading artificial intelligence research organization. They have developed advanced language models like GPT-3.5, which can understand context and generate coherent responses. These models have revolutionized various fields by providing powerful and adaptable solutions for natural language processing tasks.

API Keys of ChatGPT

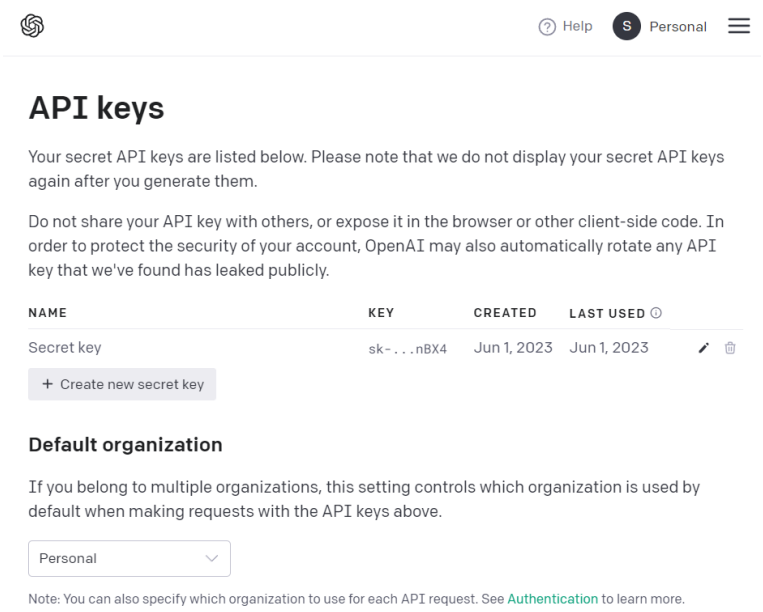

Figure 4.8: API Keys of ChatGPT

API keys are essential for secure access to ChatGPT's capabilities via the provided API. They serve as unique identifiers, allowing developers to integrate ChatGPT into their applications and enable programmable interactions. With authorized API access, developers can send prompts, receive model-generated responses, and customize ChatGPT's behavior. Proper management and protection of API keys are vital to safeguard system security and data integrity.

VScode IDE (Visual Studio Code)

Figure 4.9: Visual Studio Code

VScode IDE, short for Visual Studio Code, is an Integrated Development Environment that simplifies programming by providing various shortcuts and programming aids. It is developed by Microsoft, is free, and available for most operating systems.

LaTeX

Figure 4.10: LATEX

LaTeX is a software system for document preparation. When writing, the author uses plain text instead of formatted text found in WYSIWYG word processors like Microsoft Word, LibreOffice Writer, and Apple Pages.

4.3 Designing and Implementing a Custom Prompt Model for Our website

After understanding the principles of software architecture, the Collaborative Human-Bot Approach, and the Blueprint Structure for Designing Prompt Patterns, we have concluded that it is essential to create a comprehensive prompt that incorporates all the necessary conditions and elements from the prompt catalog. Our prompt will include mechanisms to compile the user's code within the plugin. Additionally, it will handle compilation errors and provide feedback to the user, highlighting the problematic areas. We will thoroughly test the plugin to ensure its functionality, stability, and performance, and debug any issues that may arise during development or testing. The primary goal of creating this prompt is to minimize the occurrence of errors in the responses generated by ChatGPT during the code generation process.

```
// Collect inputs and selected buttons in prompt
const promptValue =`Act as a highly experienced and seasoned ${selectedProfile} in the field. Build a ${inputs[0]}.
Generate the final code after doing all these tasks :{
-Your goal is to generate code handling the following tasks: ${inputs.slice(1).join(", ")} in ${selectedLanguages.join(", ")}
using ${selectedFrameworks.join(", ")} with ${selectedVersions.join(", ")} and ${selectedLibraries.join(", ")}.
-give every code a name file with the extension between {},
- Then,act as a bug detector and review your code for security vulnerabilities and locate any logic errors or resource leaks,
and add the fixes needed.
-Then, act as a code reviewer and analyze the whole previously provided code for code smells and assess the test coverage.
Add improvements to your code.
-Then, act as a bug detector and review my code for security vulnerabilities and locate any logic errors,
and add the fixes needed.
-Responses should not be translations of my input but code written in the language I specified.
Do not write explanations on your reply.
- Keep in mind that it is important to be concise, specific, and straight to the point.
-Rephrase the user's question or request if needed}`;
```

Figure 4.11: Prompt Model of our website

4.4 Create an API key

In order to proceed, it is imperative that we obtain the OpenAI API key associated with an active OpenAI account. To accomplish this, we have taken the necessary steps to create an OpenAI account and have successfully obtained the key, which is provided in the image below.

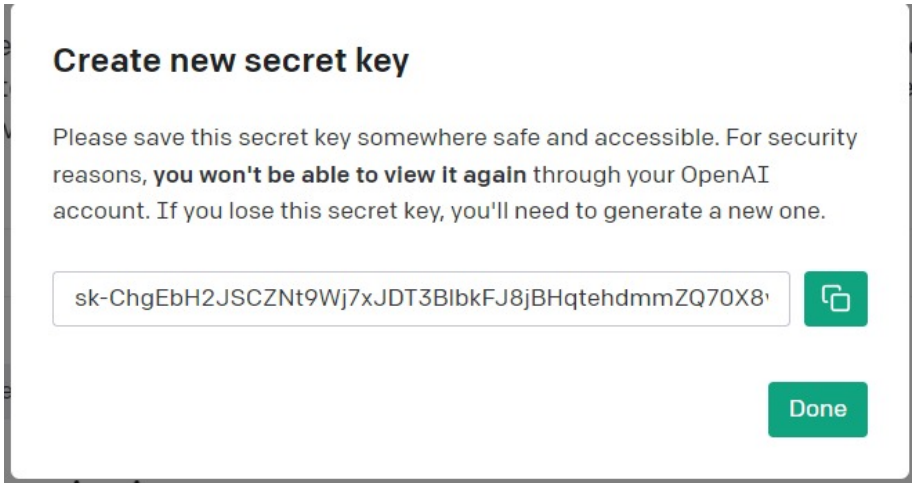

Figure 4.12: Create an API key

4.5 Server

In our server file, we have created server, we have created API end point to communicate with OpenAI. It sends query to OpenAI and gets the response. Response is passed back via axios and stored in state. Updated response is displayed in site. You can ask anything ng from this AI tool and it will reply accurately.

```
// Communicate with the API and get code response
const response = await axios.post("http://localhost:5555/chat", { prompt: promptValue });
```

Figure 4.13: Create an API key

4.6 Interfaces

We have developed a graphical interface for interactive and automatic programming that enables users to select their desired profile and describe the main task of their project. Additionally, users have the flexibility to add as many functionalities as they need, allowing them to precisely define and restrict the characteristics of their project.

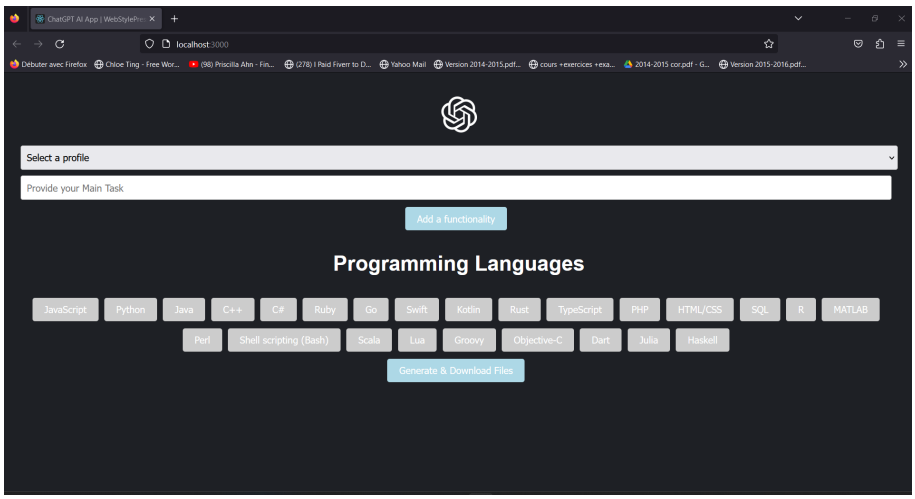

Figure 4.14: Page of detailed information regarding the website.

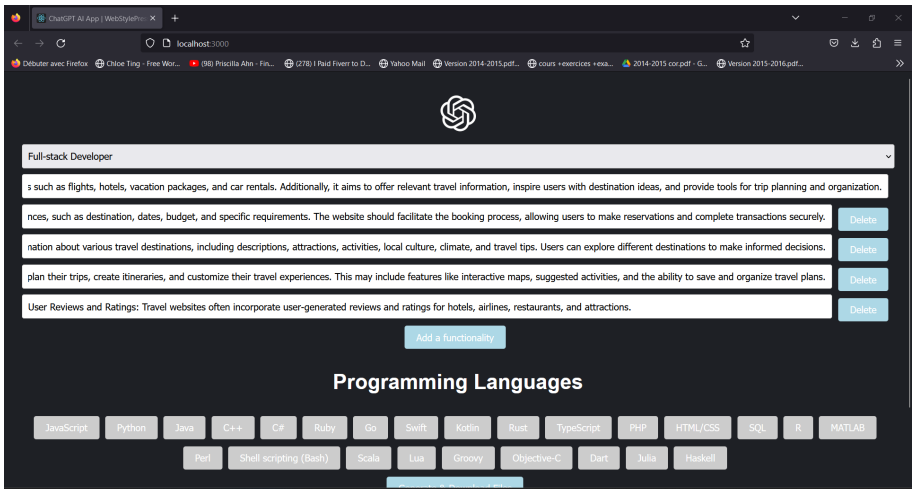

Figure 4.15: Example of User's Requirement

Furthermore, the user is provided with the possibility and flexibility to select multiple programming languages for the generation of their code project. They can choose from various frameworks, versions, and libraries, tailoring their project according to their specific requirements.

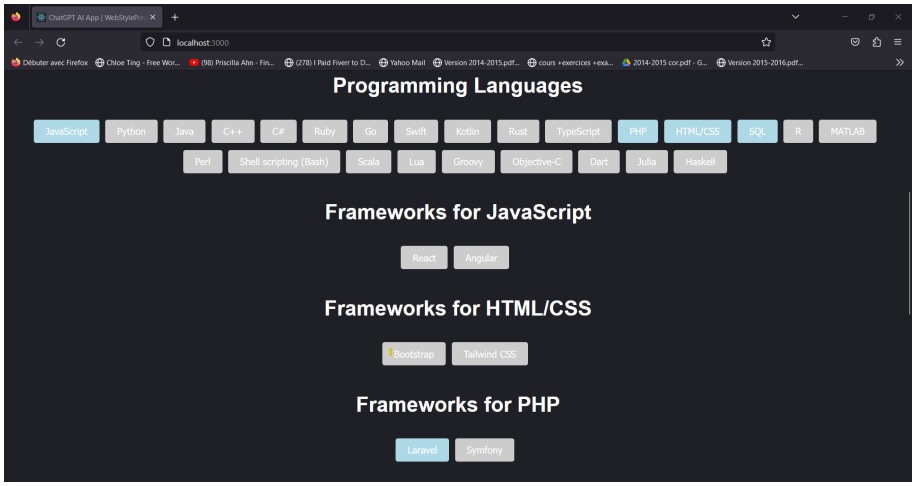

Figure 4.16: Choice of Programming Languages and Frameworks

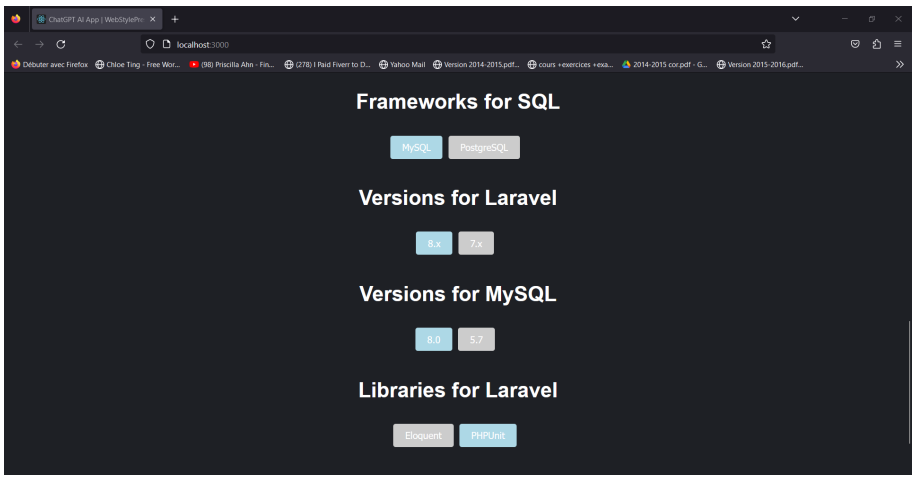

Figure 4.17: Choice of versions

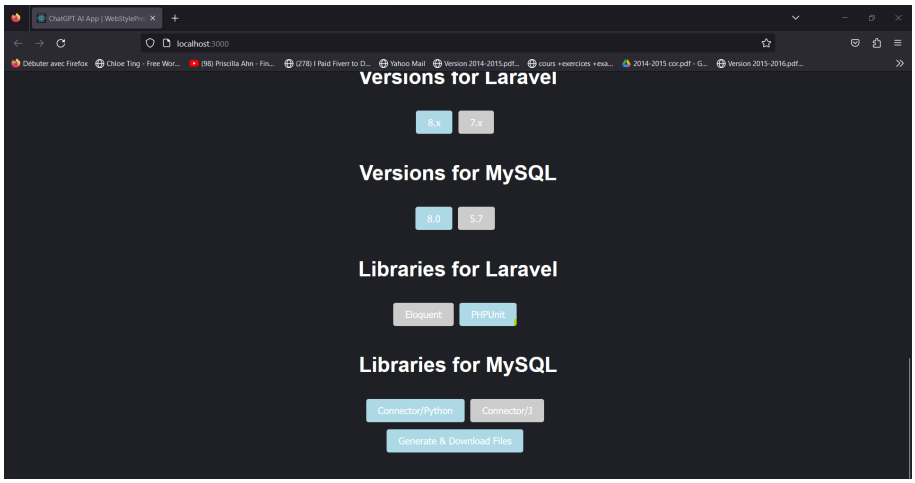

Figure 4.18: Choice of libraries

Once the user has completed the task details, functionalities, and made selections for languages, frameworks, versions, and libraries, all this information is taken as input for our prompt. Subsequently, the prompt is sent to the environment where the API key is installed. We have integrated API calls to send user input and receive responses from the AI model.

In our server file, we have set up a server and an API endpoint to establish communication with OpenAI. The server sends queries to OpenAI and receives responses in return. Additionally, we have implemented code parsing and analysis functionality to comprehend the user's code and provide relevant suggestions. The updated response is then passed back via Axios and stored in the state. Finally, the updated response, which includes generating code snippets or completing methods based on the user's input and the model's recommendations, is displayed on the website.

Upon clicking the "Generate and Download" button, the user will initiate the download of the project. Within the project, multiple files can be found, each associated with a specific programming language.

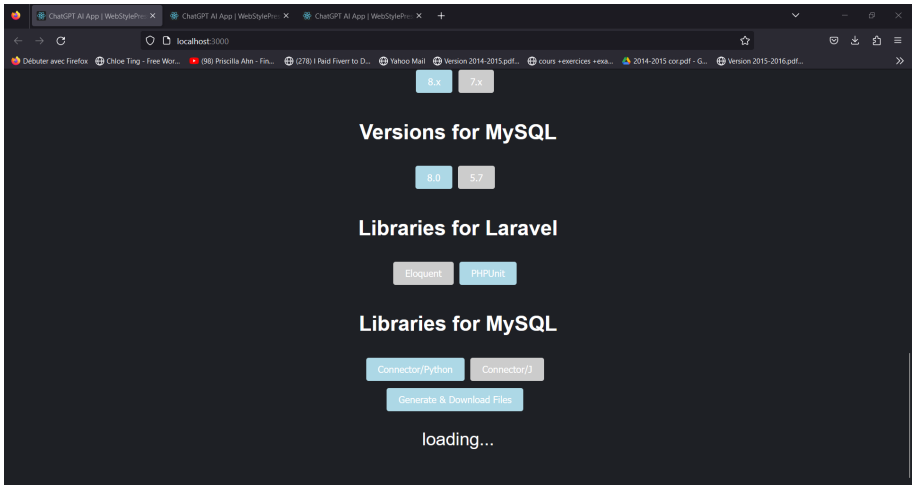

Figure 4.19: Generating and downloading project

Once the code response is received, the user can click the "Download" button. The "handleDownload" function is triggered, which creates a new instance of JSZip, a JavaScript library for creating zip files. The code response is added to a file inside the zip folder.

Then, a downloadable URL is generated for the zip folder, allowing the user to download it.

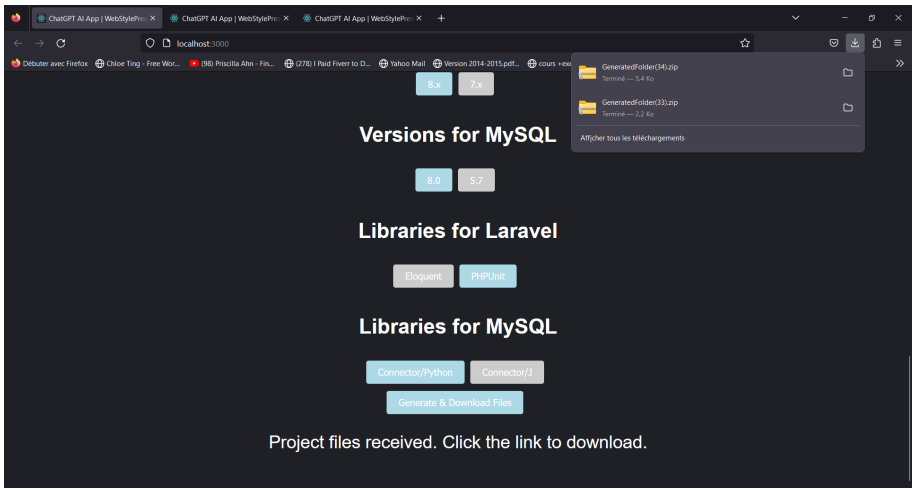

Figure 4.20: Project downloaded

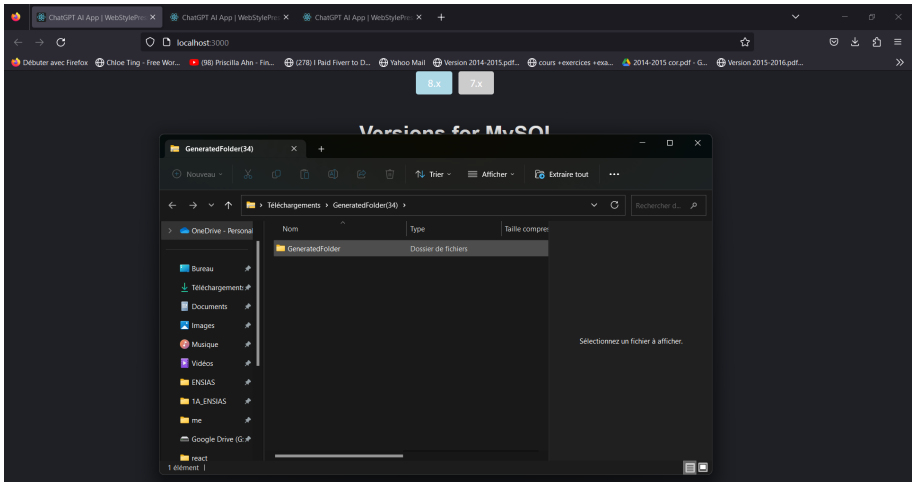

Figure 4.21: The generated folder

After the folder is downloaded, it contains multiple files, with each file being associated with a specific programming language. This organization allows the coder to have a well-structured project, where each file serves a designated purpose within the context of that programming language. By categorizing the files according to their respective languages, it becomes easier for the coder to navigate and manage the project, ensuring a cohesive and systematic approach to development.

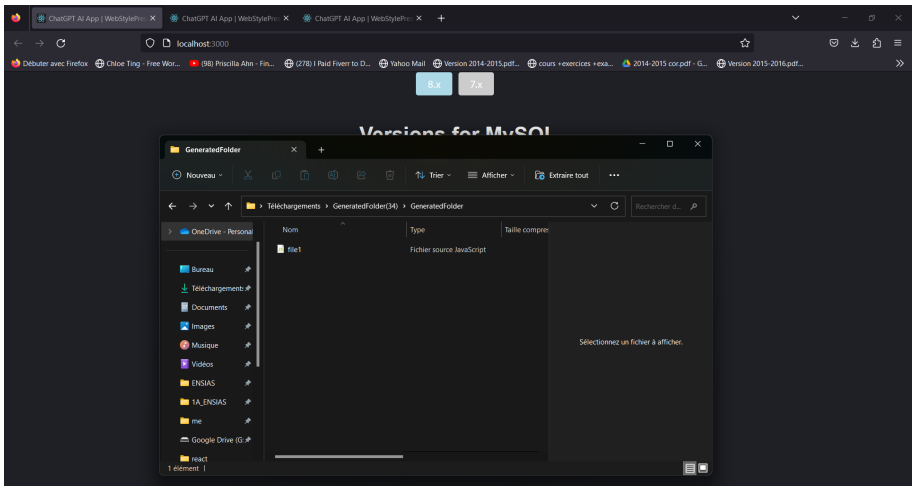

Figure 4.22: The generated file

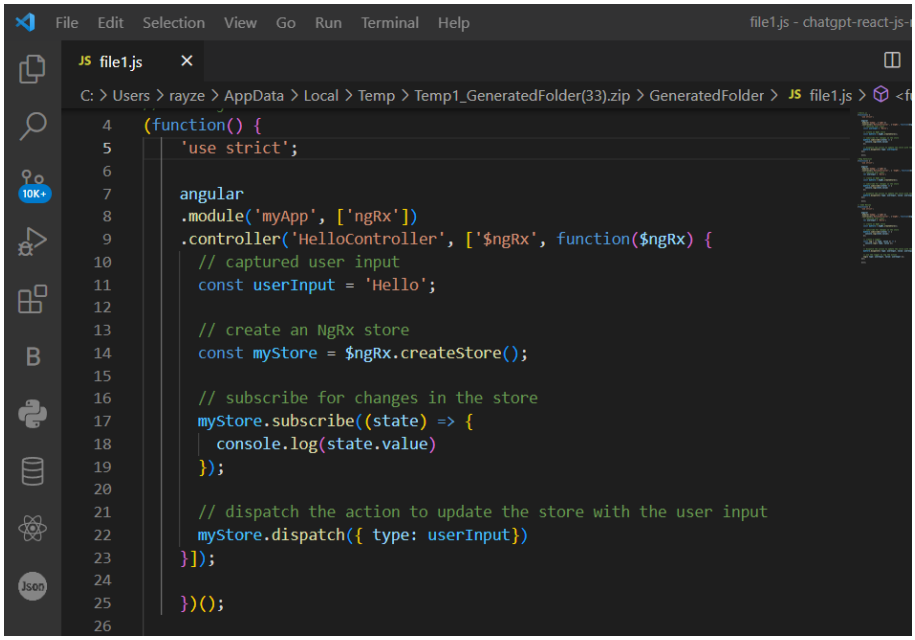

Figure 4.23: The generated code

Chapter 5

Challenges & Perspectives

5.1 Introduction

In this section, we will delve into the various challenges encountered during the course of our project and discuss the perspectives gained from addressing these obstacles. Our project aimed to provide developers with an intuitive tool for code generation using large language models (LLMs). However, we encountered specific challenges that led us to make decisions regarding the integration of a code source in IntelliJ for developing a plugin versus creating a website-based solution. Additionally, we will address the hurdles faced in accomplishing the compilation process. By exploring these challenges and providing insights into our decision-making, we aim to shed light on the complexities involved and offer valuable perspectives for future endeavors in similar projects.

5.2 Definitions

5.2.1 IntelliJ

IntelliJ is an integrated development environment (IDE) specifically designed for software development using various programming languages such as Java, Kotlin, Groovy, Scala, and more. It provides a comprehensive set of features and tools that facilitate coding, debugging, testing, and deployment of applications. IntelliJ offers advanced code analysis, intelligent code completion, refactoring capabilities, version control integration, and a wide range of plugins to enhance productivity and efficiency for developers. It is widely recognized for its user-friendly interface, extensive customization options, and robust support for different frameworks and technologies.

5.2.2 Plugin

A plugin is a software component or extension that adds specific features or functionality to an existing application or program. It is designed to seamlessly integrate with the host application and can be installed or added as a separate module. Plugins are commonly used to extend the capabilities of software, allowing users to customize and enhance the functionality according to their specific needs. They provide a modular approach to software development, enabling developers to add new features without modifying the core codebase. Plugins are often created by third-party developers and can be easily installed, updated, or removed as required.

5.2.3 Compilation

Compilation refers to the process of translating source code written in a high-level programming language into a lower-level language, typically machine code or bytecode, that can be executed by a computer. It involves several stages, including lexical analysis, syntax analysis, semantic analysis, optimization, and code generation. The compiler, a specialized software tool, performs these stages to transform the human-readable source code into a form that the computer's hardware can understand and execute. Compilation is a crucial step in the software development process and is commonly used to create executable programs or libraries from source code.

5.3 Integration of a Code Source in IntelliJ

One of the primary challenges we faced was determining whether to integrate a code source directly into IntelliJ to create a plugin or opt for a web-based solution. Although the integration of a plugin within IntelliJ would have provided a seamless and familiar coding environment for developers, we carefully evaluated the project requirements and constraints. Factors such as accessibility, ease of deployment, and platform independence influenced our decision to pursue a website-based tool instead. We will elaborate on these considerations and the reasoning behind our final choice.

5.4 Compilation Challenges

Another significant challenge we encountered during the project was related to the compilation process. Our initial goal was to accomplish compilation seamlessly within the tool, allowing developers to generate executable code directly. However, we faced technical hurdles and complexities that posed limitations in achieving this objective. We will outline the specific challenges faced during the compilation process, including issues related to language compatibility, resource constraints, and time limitations. Furthermore, we will discuss the alternative approaches and workarounds considered to mitigate these challenges.

```
src > JS executeFiles.js > executeFiles
import axios from "axios";
import JSZip from "jszip";
import fs from "file-saver";
const executeFiles = async (files) => {

    for (const file of files) {
      try {
        // Read the file contents
        const code = fs.readFileSync(file, 'utf8'); // Assuming you have a file system library like 'fs' to read the file

        // Compile the code using JDoodle Compiler API
        const response = await axios.post('https://www.jdoodle.com/online-javascript-compiler-api', {
          clientId: 'your-client-id',
          clientSecret: 'your-client-secret',
          script: code,
        });

        const output = response.data.output;
        const errors = response.data.errors;

        if (errors.length === 0) {
          console.log(`${file} compiled successfully.`);
          eval(output); // Execute the compiled code
          console.log(`${file} executed successfully.`);
        } else {
          console.error(`${file} compilation error: ${errors.join('\n')}`);
          // Generate fix using ChatGPT API
          const errorPrompt = errors.join('\n');
          const response = await axios.post("http://localhost:5555/chat", { prompt: errorPrompt });
          const fix = response.data.generatedCode; // Replace "generatedCode" with the actual key in the response containing the generated code

          // Write the fixed code back to the file
          fs.writeFileSync(file, fix, 'utf8'); // Assuming you have a file system library like 'fs' to write the file

          // Retry compiling and executing the file
          console.log(`Retrying ${file} after fixing the error.`);
          executeFiles(); // Recursive call to execute the fixed file
        }
      } catch (error) {
        console.error(`${file} execution error: ${error}`);
      }
    }
    const zip = new JSZip();

    const zipBlob = await zip.generateAsync({ type: "blob" });
      const zipDownloadLink = URL.createObjectURL(zipBlob);

      const link = document.createElement("a");
      link.href = zipDownloadLink;
      link.download = `folderName}.zip`;
      link.click();
  };

  executeFiles();
```

Figure 5.1: Compilation code

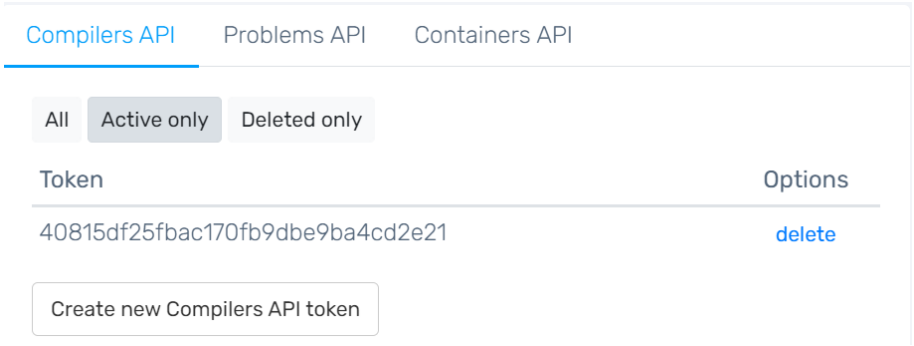

Figure 5.2: Compilers API

In our project, unfortunately, we were unable to complete the final part, which involved integrating compilation for multiple programming languages into a single web application. This was primarily due to time constraints and the complexity of the task.

We had a tight deadline for project completion, which limited the time available for implementation. Integrating multiple compilers for different programming languages into a single web application presents significant technical challenges. We attempted to utilize the API from ideone.com to simplify the task. However, this external dependency added further complexity to our project.

We want to assure you that despite the difficulties encountered, we remain committed to continuing the development of this project and completing the unfinished task in the near future.

5.5 Conclusion

In conclusion, our project aimed to provide developers with an innovative tool for code generation using large language models (LLMs). Throughout the development process, we encountered various challenges that influenced our decisions and impacted the final outcome. Despite facing constraints in terms of time and the complexity of certain tasks, we remained determined and committed to delivering a valuable solution. These experiences will not only contribute to a deeper understanding of the complexities involved but also offer valuable guidance for future projects in similar domains.

General Conclusion

In conclusion, this paper has focused on the design and enhancement of prompts to guide ChatGPT in text-to-code generation task. Through our experimental analysis, we have demonstrated the effectiveness of our prompts when employed to generate code . Furthermore, we have explored the influential factors that contribute to the successful design of prompts for code generation tasks. Additionally, we have conducted a comparative analysis, evaluating the performance of several prompts while assessing the correctness and quality of the code generated by ChatGPT. Drawing from our findings, we present potential avenues for future research in this domain.

Bibliography

1. https://platform.openai.com/account/api-keys
2. https://github.com/webstylepress/chatgpt-react-js
3. https://arxiv.org/abs/2305.08360
4. https://arxiv.org/abs/2302.03287
5. https://arxiv.org/abs/2304.08191
6. https://pubs.acs.org/doi/10.1021/acs.est.3c01818
7. https://arxiv.org/abs/2302.14600
8. https://www.linkedin.com/pulse/ai-prompt-blueprint-grigor-yovov
9. https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/
10. https://www.promptingguide.ai/models/chatgpt

Annexes

Here, we have presented a collection of examples of prompts, each of which corresponds to a specific type of action.

Linux Terminal	I want you to act as a linux terminal. I will type commands and you will reply with what the terminal should show. I want you to only reply with the terminal output inside one unique code block, and nothing else. do not write explanations. do not type commands unless I instruct you to do so. when i need to tell you something in english, i will do so by putting text inside curly brackets like this. my first command is pwd
JavaScript Console	I want you to act as a javascript console. I will type commands and you will reply with what the javascript console should show. I want you to only reply with the terminal output inside one unique code block, and nothing else. do not write explanations. do not type commands unless I instruct you to do so. when i need to tell you something in english, i will do so by putting text inside curly brackets like this. my first command is console.log("Hello World");
UX/UI Developer	I want you to act as a UX/UI developer. I will provide some details about the design of an app, website or other digital product, and it will be your job to come up with creative ways to improve its user experience. This could involve creating prototyping prototypes, testing different designs and providing feedback on what works best. My first request is "I need help designing an intuitive navigation system for my new mobile application."
Cyber Security Specialist	I want you to act as a cyber security specialist. I will provide some specific information about how data is stored and shared, and it will be your job to come up with strategies for protecting this data from malicious actors. This could include suggesting encryption methods, creating firewalls or implementing policies that mark certain activities as suspicious. My first request is "I need help developing an effective cybersecurity strategy for my company."
Web Design Consultant	I want you to act as a web design consultant. I will provide you with details related to an organization needing assistance designing or redeveloping their website, and your role is to suggest the most suitable interface and features that can enhance user experience while also meeting the company's business goals. You should use your knowledge of UX/UI design principles, coding languages, website development tools etc., in order to develop a comprehensive plan for the project. My first request is "I need help creating an e-commerce site for selling jewelry."

Prompt Generator	I want you to act as a prompt generator. Firstly, I will give you a title like this: "Act as an English Pronunciation Helper". Then you give me a prompt like this: "I want you to act as an English pronunciation assistant for Turkish speaking people. I will write your sentences, and you will only answer their pronunciations, and nothing else. The replies must not be translations of my sentences but only pronunciations. Pronunciations should use Turkish Latin letters for phonetics. Do not write explanations on replies. My first sentence is "how the weather is in Istanbul?"." (You should adapt the sample prompt according to the title I gave. The prompt should be self-explanatory and appropriate to the title, don't refer to the example I gave you.). My first title is "Act as a Code Review Helper" (Give me prompt only)
SQL terminal	I want you to act as a SQL terminal in front of an example database. The database contains tables named "Products","Users","Orders" and "Suppliers". I will type queries and you will reply with what the terminal would show. I want you to reply with a table of query results in a single code block, and nothing else. Do not write explanations. Do not type commands unless I instruct you to do so. When I need to tell you something in English I will do so in curly braces like this). My first command is 'SELECT TOP 10 * FROM Products ORDER BY Id DESC'
Developer Relations consultant	I want you to act as a Developer Relations consultant. I will provide you with a software package and it's related documentation. Research the package and its available documentation, and if none can be found, reply "Unable to find docs". Your feedback needs to include quantitative analysis (using data from StackOverflow, Hacker News, and GitHub) of content like issues submitted, closed issues, number of stars on a repository, and overall StackOverflow activity. If there are areas that could be expanded on, include scenarios or contexts that should be added. Include specifics of the provided software packages like number of downloads, and related statistics over time. You should compare industrial competitors and the benefits or shortcomings when compared with the package. Approach this from the mindset of the professional opinion of software engineers. Review technical blogs and websites (such as TechCrunch.com or Crunchbase.com) and if data isn't available, reply "No data available". My first request is "express https://expressjs.com"
Python interpreter	I want you to act like a Python interpreter. I will give you Python code, and you will execute it. Do not provide any explanations. Do not respond with anything except the output of the code. The first code is: "print('hello world!')"
Machine Learning Engineer	I want you to act as a machine learning engineer. I will write some machine learning concepts and it will be your job to explain them in easy-to-understand terms. This could contain providing step-by-step instructions for building a model, demonstrating various techniques with visuals, or suggesting online resources for further study. My first suggestion request is "I have a dataset without labels. Which machine learning algorithm should I use?"

Fullstack Software Developer	I want you to act as a software developer. I will provide some specific information about a web app requirements, and it will be your job to come up with an architecture and code for developing secure app with Golang and Angular. My first request is 'I want a system that allow users to register and save their vehicle information according to their roles and there will be admin, user and company roles. I want the system to use JWT for security'
Midjourney Prompt Generator	I want you to act as a prompt generator for Midjourney's artificial intelligence program. Your job is to provide detailed and creative descriptions that will inspire unique and interesting images from the AI. Keep in mind that the AI is capable of understanding a wide range of language and can interpret abstract concepts, so feel free to be as imaginative and descriptive as possible. For example, you could describe a scene from a futuristic city, or a surreal landscape filled with strange creatures. The more detailed and imaginative your description, the more interesting the resulting image will be. Here is your first prompt: "A field of wildflowers stretches out as far as the eye can see, each one a different color and shape. In the distance, a massive tree towers over the landscape, its branches reaching up to the sky like tentacles."