The use of large language models in reverse engineering class diagrams

Name of student 1: Victor Campanello
Name of student 2: Shariq Shahbaz
Proposed academic supervisor's name: Daniel Strüber & Vladislav Indykov
Have your proposed academic supervisor clearly stated that he/she will supervise you: YES
Will the thesis work be conducted in collaboration with an external organization: NO

I. Introduction

Class diagrams are an important tool for the understanding of software projects. They provide an abstract overview of projects and can therefore be used as an on-boarding tool for new developers. However, a substantial problem arises due to the low frequency at which the diagrams are updated [1]. This can cause a roadblock, since the efficacy of the diagram in helping with the understanding of a particular project relies on the diagram being an accurate representation of the codebase. A solution to this dilemma would be to lower the time investment required to create and maintain the class diagram, such as with the use of specific reverse engineering tools which can generate class diagram from inputted code. The problem is that these tool are too literal as they lack the ability to perform abstractions [2], causing the diagrams to look cluttered. An improvement of these reversed engineered class diagrams was made by Osman et al. [3] and Thung et al. [4] which both attempted to condense the reversed engineered diagrams. Osman et al. [3] and Thung et al. [4] use traditional machine learning techniques to abstract away classes in order to improve readability, as not all classes are needed to understand a codebase. Further improvements were made upon the work of Osman et al. and Thung et al. by Yang et al. who reduced the needed for manual labeling when using machine learning algorithms for condensing reverse engineered class diagrams [5].

Another way to tackle the problem of easily creating class diagrams is with the use of Large Language Models (LLMs). There has been a significant increase in the complexity and availability of LLMs over the last few years [6]. These LLMs have an edge over traditional machine learning techniques that are restricted to specific sizes, dimensions and modes of inputs and outputs, as explored by Osman et al. [3] and Thung et al. [4]. The ability of modern LLMs to take intake large amounts of multi-modal data and produce useful output makes them a valuable tool for various industries, in particular software development and documentation. In order for LLMs to provide utility within the realm of software documentation, it is important to understand their handling of large code bases as input and their ability to grasp the various complex relationships among the code. This understanding in particular is of the utmost importance, if these LLMs are to be tweaked for software documentation specific use cases. Such is the purpose of this paper: to analyze, organize and fine tune the output from a LLM to produce class diagrams in an attempt to aid the streamlining of the software documentation process.

For the purposes of this research, we have decided to use GPT-4 as it is one of the best available LLMs [7] [8]. GPT-4 is a multi-modal LLM that outperforms a variety of other LLMs on the metrics of hallucination and common-sense reasoning, as found by Minaee et al. [8], making it a good fit for this research.

The research process is divided into two phases. The first phase will put GPT-4's code interpretation abilities to the test, as the model is used to generate class diagrams using code as input. The output from GPT-4 will be compared to human-generated diagrams. Building upon the insights gathered in phase one, in phase two we will once again generate class diagrams, but this time include the *Human Abstraction Framework* in the prompt as well, in an attempt to imitate the human abstractions detailed by Zhang et al. [9]. This research aims to explore the following questions:

- RQ1) How effectively does GPT-4 abstract code relationships when generating class diagram on its own, as compared to the human-generated diagrams?
- RQ2) How effectively does GPT-4 abstract code relationships when generating class diagrams, utilizing the human abstraction framework as input along with code input?
- RQ3) How does the inclusion of the human abstraction framework within the prompt affect the outputted diagrams, as compared to when it is excluded?

In order to address the research questions and evaluate GPT-4's class diagram generating abilities, we first need to conduct some *pre-experiments*. As there are several ways a file can be be provided to GPT4, e.g the files can be attached with the prompt or their content can be a component of the prompt. We must first determine the best approach for file handling in order for GPT-4 to have optimal comprehension of the codebase. Furthermore, as the prompt to a large extent determines the output quality of the LLM [10], it is then critical to have a fine-tuned prompt that allows GPT-4 to best generate class diagrams. The aforementioned experiments will provide us pre-requisite knowledge, improving the overall output from the actual experiments concerning RQ1 and RQ2.

The research questions will be answered via a controlled experiment, comparing the performance of GPT-4 in creating class diagram with human abstractions with a set of five class diagrams created by five development teams. The answers to these questions will act as a stepping stone for future research to use these rapidly improving LLMs in order to streamline the software documentation process and aid in the on-boarding of new developers with the assurance that the software models are accurate and up to date.

II. Related Work

A. Using machine learning for reverse engineering of class diagrams

A solution to automatically generate class diagrams is the use of machine learning to do reverse engineering. Machine learning is broadly defined as the process of using past data to create an algorithm to predict future data [11]. On the other hand, reverse engineering is the process of creating a representation of an already existing system [12]. In this case this would entail automatically creating class diagrams from source code using the source code as input. This was the purpose of Osman et al. [3] and Thung et al. [4] work. By using auto-generated class diagrams in combination with machine learning algorithms, the papers of Osman et al. and Thung et al. attempt to make the class diagram more insightful. However, one lacking aspect is that the exact changes made by the algorithms are not categorized, i.e the specific changes are not organized as a framework such as the one presented by Zhang et al. [9]. This does not, however, take away from the utility of the papers to the research at hand.

Another important contribution comes from Yang et al. [5]. Their paper focuses on the labelling of a dataset, documenting techniques that be can utilized by future research in order to improve the labelling process of reversed engineered class diagrams. As was done by Zhang et al. in the implementation of their framework [9]. Although the goals of this research do not involve the labeling of diagrams, the contents of this paper may still prove useful for designing the specific prompts that will eventually be passed to the LLM in question.

The papers discussed above provide useful insight into the problem at hand. However, their approaches to the problem vary significantly from the approach taken in this research. This is mainly due to the fact that streamlined public access to advanced Large Language Models is a recent development. As large language models have the capability to far outperform traditional machine learning algorithms [6], it is then critical to evaluate their performance for the given problem. This has not yet been done, and hence is the purpose of this research.

B. Background

1) The Human Abstraction Framework: A critical paper in the understanding of the development of LLM-generated class diagrams is Zhang et al [9]. They looked at the different ways developers abstract their code as they create class diagrams. Zhang et al. hand picked a set of five software projects with class diagrams from the *Lindholmen Dataset* [13] and performed detailed breakdowns of all abstractions therein by manually matching 466 classes, 1352 attributes, and 2634 operations from source code to 338 model elements [9]. Zhang et al. broadly categorized differences between source code and diagram into a taxonomy of Real abstractions, Disagreements and Inaccuracies. We will call this the *Human Abstraction Framework*. These three categories are defined in Table I with a detailed definition of all Real abstractions in Table II. The need for a categorized framework is vital

TABLE I
Definitions of key terms, as per Zhang et al. [9]

Real Abstraction	Cases where the model uses elements that specify more general semantics or contain fewer details than what can be found in the source code
Disagreements	Cases where the model uses elements that specify more specific semantics or contain more or different details than what can be found in the source code
Inaccuracies	Differences that cannot be classified as a difference in level of specificity and detail, but rather as non- conceptual differences in representation

in establishing what steps needs to followed to make sure the LLM is making the correct decisions when it excludes information from it's created class diagrams. This is due to the fact that some information is critical and should not be abstracted away while some other information can be removed or simplified to improve understanding of a particular diagram. For example the existence of helper methods, utility classes, library implementations and sets of methods with similar functionality in a class diagram contributes little to the overall understanding of the system. Hence, if no abstractions are made then the diagrams would become too cluttered rendering them no different than traditional reversed engineered class diagrams. This would also prove inferior to the work of Osman et al. [3] and Thung et al. [4] which can successfully condense the information of the class diagrams.

2) Large Language Models and PlantUML: There is a vast variety of large deep learning models available. We can classify them into three categories based on the types of outputs they produce. Image generating models, such as DALL-E and Stable Diffusion, output images given text as input. Newer video generating models such as Sora, have the ability to generate videos using text, but given its multimodal nature it is also capable of taking in images and videos as input [14].

Text generating models, such as Gemini and GPT, can take in a variety of different kinds of input. They are able to use *Natural Language Processing* (NLP) techniques to not only understand but also output text in Natural Language [7]. However, as these models are limited to text as their output and the goal of this research is to create diagram images, a syntactical solution is required. PlantUML is our solution of

TABLE II
CASES OF REAL ABSTRACTION, AS PER ZHANG ET AL. [9]

Subsystems
Subsystem omission: The model focuses on one or more sub- systems of the system only and omits all information about other parts of the code.
Classes
Inheritance structure omission: Inheritance structures in the source code are not shown in the model
Class omission: Classes present in the source code of the modeled system part are not shown in the model
Class summary: Two or more classes present in the source code are shown as one class in the model
Attributes
Attribute omission: Attributes in the source code are not shown in the model
Attribute summary: Multiple attributes in the source code are shown as one attribute in the model.
Attribute type omission: The type of an attribute in the source code is not shown in the model.
Default value omission: An attribute in the source code has a default value that is not shown in the model.
Operations
Operation omission: Operations in the source code are not shown in the model.
Operation summary: Multiple operations in the source code are shown as one operation in the model.
Parameter omission: Parameters in the source code are not shown in the model.
Parameter name omission: Parameter names in the source code are not shown in the model.
Return type omission: The return type of a method in the source code is not shown in the model.
Collection type underspecification: Either the types of objects that can be stored in collections as specified in the source code are not shown in the model, which only shows the type of the collection, or only the types of objects are shown, but not the information that there is a collection of these objects.
Relationships (between classifiers)
Relationship omission: Relationships in the source code are not shown in the model.
Relationship loosening: An attribute (i.e. owned element) in the source code is modeled as a named association in the model (and not as a composition or aggregation).
Relationship summary: For two classes that access each others' values indirectly via a third class in the source code a direct association is shown in the model.

choice [15], which provides a text-based approach to creating UML diagrams by following a specific syntax. The LLM can be instructed to provide its output in the PlantUML syntax [15], which can easily be converted into class diagram images through PlantText [16].

III. RESEARCH METHODOLOGY

This study aims to perform a controlled experiment in order to evaluate the efficacy of using GPT-4 for generating class diagrams. Two pre-experiments will also be performed in order to determine an effective prompt and the best approach to file handling for GPT-4 which will then be used in the controlled experiment. The controlled experiment utilizes the original diagram creators as the control group, the mode of creation (manual vs LLM) as the independent variable and the dependent variable is the performance as compared to the human-generated diagrams.

The human-generated diagrams are assumed to be the ground truth, i.e well abstracted class diagrams that provide the best possible overview for the system given the codebase. Therefore, the data analysis mainly involves comparing the LLM-generated diagrams to the human-generated diagrams, on the metrics of *Correctness Ratio (CR)* and *Mean Rate of Abstraction (MRA)*.

We will perform a statistical analysis on the results from RQ1 and RQ2 in order to then answer RQ3. For this we will use hypothesis testing, and in the case that the null hypothesis can be rejected, we will use an effect size test (A12 from Vargha and Delaney [17]) to determine the magnitude of the differences between the two results.

The justification for the project and LLM selection, as well as a detailed description of the pre-experiment, data collection and analysis is described below.

A. Project Selection

To evaluate the capability of GPT-4 to create real abstractions over class diagrams, existing software projects with class diagrams are required. These project can then be given to GPT-4 as input and their resulting class diagrams can be compared to the existing one.

This research will utilize the same five projects as Zhang et al [9]. Since the *Human Abstraction Framework* is crucial to this project and the framework has only been analyzed on these projects, the same projects were chosen. Doing this breakdown ourselves does not fit within the scope of our research. This is due to the fact that performing this analysis on new projects is very time consuming as evidenced by the fact Zhang et al. manually matched 466 classes, 1352 attributes, and 2634 operations from source code to 338 model elements, all of which took one year of work [9]. The aim is to extend the work done by Zhang et al. by comparing their chosen class diagrams with the ones produced by GPT-4. This allows for a one to one comparison between human-generated class diagrams and GPT-4-generated class diagrams using the same projects as input.

B. Large Language Model Selection

Provided the information from *Large Language Models and PlantUML* as discussed in Section II-B2, we decided that Image and text generating models are the most relevant for the goals of this research. Another important factor in picking an LLM, is the *token limit*. Tokenization is the process of "dividing up the input text, which to a computer is just one long string of characters, into sub-units, called *tokens*", as per Grefenstette [18]. Tokenization is how NLP models process their input [19], but each model is limited by the number of tokens it can process, which limits how large of an input the LLM can accept in the prompt. We have eliminated the possibility of using a deep learning text to image model, such as Dall-E. The reason for this is that these models have a significantly low token limit [20]. The low token limit on these models makes it impossible to use a codebase as input, as the amount of tokens in the chosen projects far exceeds the input capabilities of these models.

A consequence of this choice is that it leaves text to text models as the only option. There are several Large Language Models available that can be used for evaluating the research questions, the most prominent ones include Claude, Gemini, Llama and GPT-4 [8]. The following criteria were analyzed when picking the final LLM for this research.

Firstly, *Hallucinations* are an important metric for the evaluation of Large Language Models, as they can be a measure of the factual correctness of the output [21]. Within the context of LLMs, a hallucination is defined as "the generated content that is nonsensical or unfaithful to the provided source content" [22]. According to the Hughes Hallucination Evaluation Model, GPT-4 has the lowest hallucination rate, leading to the highest factual consistency [23]. Another important metric is *commonsense reasoning*, which is defined as the "ability of the model to use prior knowledge in combination with reasoning skills" and is very important in determining an LLM's capability to reason [8]. A dataset consisting of 70,000 multi-choice questions called HellaSwag ranked GPT-4 the highest at 95.3 percent correct, 6.44 points above second place [8].

Provided the above reasoning in combination with the results from Minaee et al. [8], we decided to use GPT-4 for our research. This is because GPT-4 is not only well balanced in its reasoning but also hallucinates far less often in comparison with its competitors, hence we determined it to be the best fit for generating abstracted class diagrams.

C. Pre-experiments

As aforementioned, some questions about the chosen LLM must be addressed before the final research can be conducted. Mainly, file handling, i.e the best approach for providing file content to GPT-4, as well constructing an effective prompt for class diagram generation.

1) File handling: Being a multi-modal LLM, GPT-4 is able to consume files as input [14]. It is possible to attach a maximum of 20 files along with a given prompt. The limit itself is not a problem, as the files can be batched and the results aggregated. However, there may be additional approaches for file handling with GPT-4 that offer increased comprehension. Another approach is to provide the file contents within the prompt, this approach is not equivalently applicable to media files such as images, however in this case the files in question are java files. Contemporary literature does not analyze the difference in GPT-4's comprehension given that the same input is provided through different means. This is a prerequisite for the controlled experiment, as the experiment involves providing a large number of files to GPT-4. Therefore, we must first determine the best way to provide file content to GPT-4 through a pre-experiment.

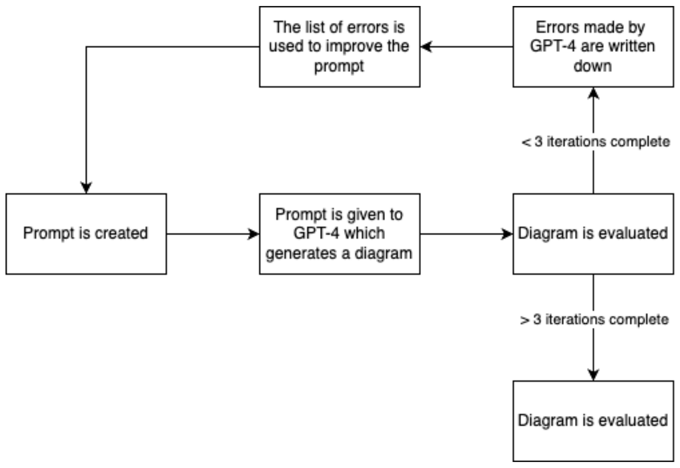
The list of errors is used to improve the prompt
Errors made by GPT-4 are written down
< 3 iterations complete
Prompt is created
Prompt is given to GPT-4 which generates a diagram
Diagram is evaluated
> 3 iterations complete
Diagram is evaluated

Fig. 1. Step by step visualisation of the fine-tuning of the prompts

Two of the five projects from our subject systems, will be randomly selected for this experiment. Each project will be used to generate two class diagrams through GPT-4. In one case, the files are attached along with the prompt. In the other, the files are provided as text content within the prompt. The outputted PlantUML syntax will then be processed through PlantText in order to generate diagram images [15] [16]. These images will be analyzed as per the metrics in *Data Analysis (Sect. III-E)* to determine the better approach for file handling with GPT-4. This approach will then used for providing files to GPT-4 in the controlled experiment.

2) Fine-tuning the prompt: When it comes to LLMs, the choice of prompt is a determining factor of the output quality [10]. It is crucial then, for our experiment to use good prompts. From manually inspecting the subject class diagrams, we observed that the diagrams differ in some stylistic choices. For example, developers made different decisions on whether to include all parameters in methods, with neither of these decisions being inherently more valid than the other. Since all class diagram have something mutually exclusive with all other diagrams, two prompts uniquely crafted for each diagram is required, one including the Human Abstraction Framework and one excluding it. Crafting specific prompts for each project may seem counter-intuitive as the overall goal is to help streamline the creation of class diagrams, however, a tool based on your LLM prompts could support the different stylistic choices through configuration options. After the initial prompts are specified, each prompt will be iterated upon three times, each time improving based on errors in the diagram made by GPT-4. The process is shown in fig 1.

The outputted PlantUML will be converted to diagram images through PlantText and be analyzed as per the metrics in *Data Analysis (Sect. III-E)*. The best prompt from the three, as determined by the experiment, will be used for the controlled experiment.

D. Data collection

The data collection process benefits substantially from the two pre-experiments. They provide insights on the best file handling approach and an effective prompt for generating class diagrams with GPT-4. The findings from the pre-experiments will then be used to generate class diagram for the selected projects. For RQ1, all the classes present within the project will be included in the prompt for GPT-4. The prompt will include detailed instructions for GPT-4 to generate a class diagram following the PlantUML syntax. The generated PlantUML syntax will then be processed through PlantText [16] which converts the PlantUML syntax into an image file. This will be repeated for 10 trials in order to account for variance in the results.

In order to determine the differences between human and AI generated diagrams, we decided to use an automated approach. This was done to ensure consistency and repeatability of the analysis while avoiding manual analysis which can be time-consuming and error-prone. However, in order to compare the two diagrams automatically the human diagram would first need to be recreated as PlantUML. The recreation was also used as an opportunity to correct the disagreements and inaccuracies in the diagrams denoted by Zhang et al [9]. It is important to note that the stylistic choices of the diagrams, for example the inclusion of getters/setters or the exclusion of return types, were not affected by this recreation. A python script was then used to compare the resulting human diagram to the GPT diagram [24]. As of now, the script has not yet been used to collect data for the final experiment, and has been limited to the pre-experiments.

For RQ2, the same steps will be taken, but the Human Abstraction Framework will also be included within the prompt. This process will also be repeated over 10 trials. The outputs from both RQ1 and RQ2 will then be used to gather insights into RQ3.

E. Data analysis

To answer RQ1 and RQ2, comparisons will be made between the PlantUML files of the Human and GPT diagrams. The Human diagram will be seen as ground truth and GPT-4 will be evaluated based on how closely it resembles the Human diagram.

An important consideration for the analysis is that all differences between the two diagrams should not be valued equally, for example, a difference in the visibility of an attribute is less impactful of an error as compared to a difference in the inclusion/exclusion of a particular class in the diagram. This leads us to create a set of categories to sort differences of similar importance as shown in table III.

We will calculate precision, recall and an F1 score for the results from RQ1 and RQ2. The positive and negative classes are defined below, where an item is either a class, interface, method, attribute or relationship.

Positive Class: An item is included in the diagram.

Negative Class: An item is excluded from the diagram.

TABLE III
CATEGORIES OF DIFFERENCES

Major	Classes, Relationships
Moderate	Attribute name, Method name
Minor	Visibility, Multiplicity, Attribute Type, Parameter name, Parameter Type, Method Return Type

Precision allows us to get a measure of how many of the determined positives are actual positives. It is defined as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall allows to us to get a measure of how many of the actual positives are determined positive. It is defined as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Precision is a useful measure when the costs of false positives is high, i.e. an item that must not be included is included by GPT. On the other hand, recall is a good measure when the cost of false negatives is high, i.e. an item that must be included is excluded by GPT. We have determined that the effort to correct false negatives and false positives of a class diagram is similar, hence it is important for us to find a balance between the precision and recall scores. We will use an F1 score in order to achieve this balance. The F1 score is defined as follows:

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

In order to evaluate RQ3, concerning the differences in the results between RQ1 and RQ2, a statistical analysis is required. We will test three different hypotheses, one for each category. The hypotheses can be stated as follows.

1) *Category: Major:*

$$H_0 : F1_{Major\,RQ1} = F1_{Major\,RQ2}$$

$$H_a : F1_{Major\,RQ1} \neq F1_{Major\,RQ2}$$

2) *Category: Moderate:*

$$H_0 : F1_{Moderate\,RQ1} = F1_{Moderate\,RQ2}$$

$$H_a : F1_{Moderate\,RQ1} \neq F1_{Moderate\,RQ2}$$

3) *Category: Minor:*

$$H_0 : F1_{Minor\,RQ1} = F1_{Minor\,RQ2}$$

$$H_a : F1_{Minor\,RQ1} \neq F1_{Minor\,RQ2}$$

In order to evaluate the hypotheses we will use hypothesis testing, as it determines the difference in the mean of two independent samples. If the distribution is determined to not be normal, we will use a Mann-Whitney U-test. Otherwise, we can use Student's t-test, a more powerful test that, however, assumes normally distributed data. Either test will allow us to decide whether or not the null hypothesis can be rejected. In case the null hypothesis can be rejected, we perform an effect size test in order to evaluate the magnitude of the differences between RQ1 and RQ2. We will use the A12 effect size test from Vargha and Delaney [17]. Using the effect size test, we can categorize the differences between the results from RQ1 and RQ2 into three categories. These categories include large, medium and small differences. Categorizing the differences allows us to determine the large scale impact of the inclusion of the human abstraction framework within the prompt, thus answering RQ3.

IV. Results

A. File handling

The current observations from our file-handling pre-experiment suggest that attaching files along with the prompt leads to more desirable results as compared to when the files are provided as text within the prompt. An example of this is provided through in *Fig. 2* and *Fig. 3*, which clearly demonstrate that the attachment approach yields a more comprehensible diagram. This is one of many such iterations that show the drastic differences between the two approaches to file-handling [25]. As of right now, the projects have only been analyzed manually, this is due to the fact that the automatic comparison script is a recent development, therefore it has not yet been used for the analysis [24]. The analysis will be updated with a more detailed comparison using the script in the coming weeks.

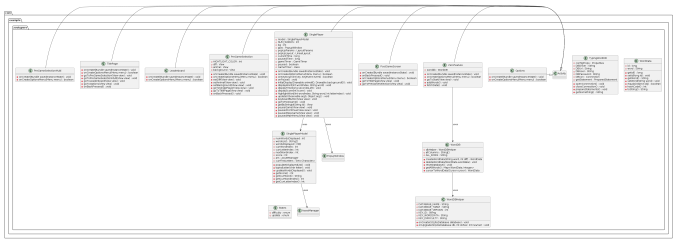

Fig. 2. Project 1 files provided as text to the prompt

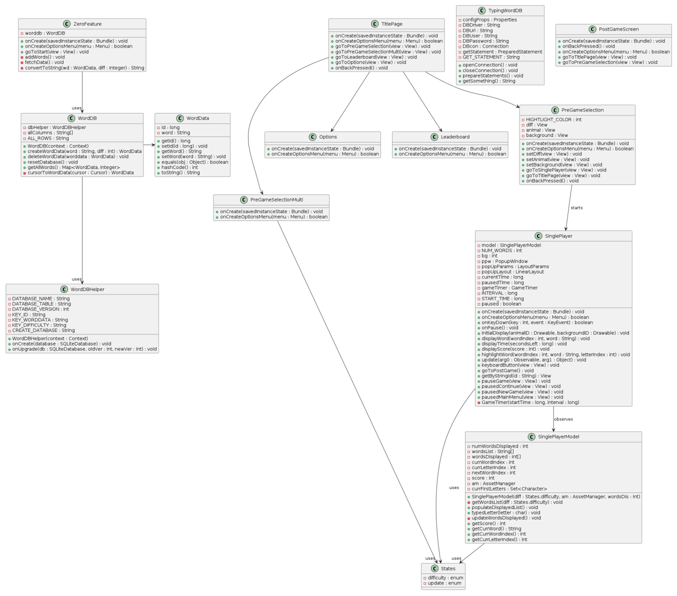

Fig. 3. Project 1 files provided as attachments to the prompt

B. Fine-tuning the prompt

As per the current observations, an iterative approach to fine-tuning the prompt leads to a progressive increase in the quality of the outputted diagram. The diagrams are analyzed, the weaknesses are found and the prompt is further tuned in an attempt to improve output quality and avoid the introduction of new problems. Figures 4, 5, 6, 7, show two examples of such progressions [25]. Similar to the analysis for the file-handling pre-experiment, the diagrams in question here were also analyzed manually but the analysis will be updated using the comparison script in the coming weeks.

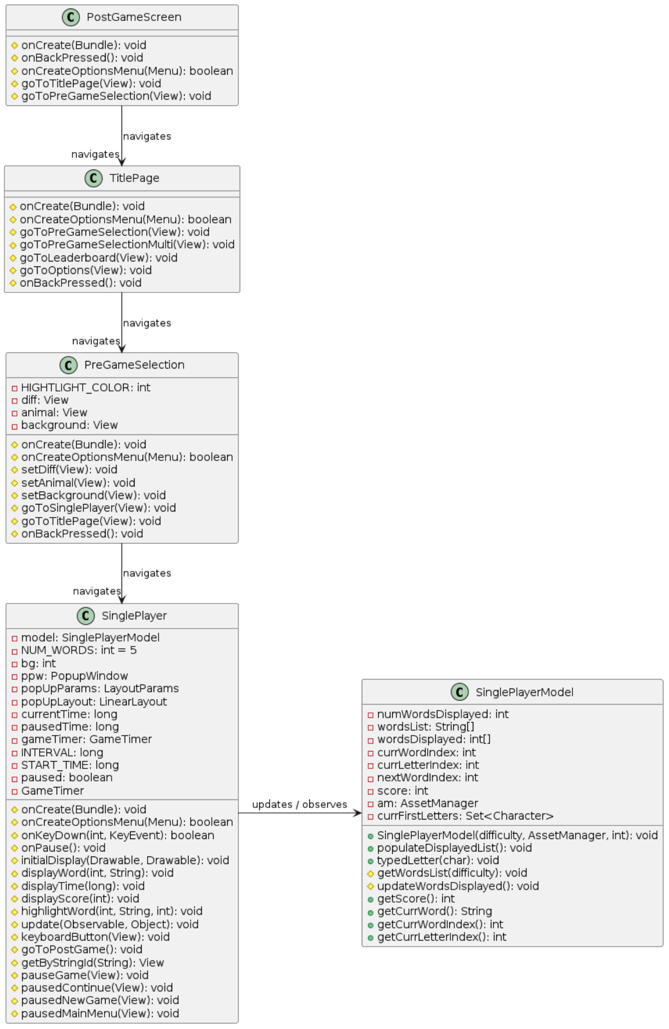
PostGameScreen
onCreate(Bundle): void
onBackPressed(): void
onCreateOptionsMenu(Menu): boolean
goToTitlePage(View): void
goToPreGameSelection(View): void
navigates
navigates
TitlePage
onCreate(Bundle): void
onCreateOptionsMenu(Menu): boolean
goToPreGameSelection(View): void
goToPreGameSelectionMulti(View): void
goToLeaderboard(View): void
goToOptions(View): void
onBackPressed(): void
navigates
navigates
PreGameSelection
HIGHTLIGHT_COLOR: int
diff: View
animal: View
background: View
onCreate(Bundle): void
onCreateOptionsMenu(Menu): boolean
setDiff(View): void
setAnimal(View): void
setBackground(View): void
goToSinglePlayer(View): void
goToTitlePage(View): void
onBackPressed(): void
navigates
navigates
SinglePlayer
model: SinglePlayerModel
NUM_WORDS: int = 5
bg: int
ppw: PopupWindow
popUpParams: LayoutParams
popUpLayout: LinearLayout
currentTime: long
pausedTime: long
gameTimer: GameTimer
INTERVAL: long
START_TIME: long
paused: boolean
GameTimer
onCreate(Bundle): void
onCreateOptionsMenu(Menu): boolean
onKeyDown(int, KeyEvent): boolean
onPause(): void
initialDisplay(Drawable, Drawable): void
displayWord(int, String): void
displayTime(long): void
displayScore(int): void
highlightWord(int, String, int): void
update(Observable, Object): void
keyboardButton(View): void
goToPostGame(): void
getByStringId(String): View
pauseGame(View): void
pausedContinue(View): void
pausedNewGame(View): void
pausedMainMenu(View): void
updates / observes
SinglePlayerModel
numWordsDisplayed: int
wordsList: String[]
wordsDisplayed: int[]
currWordIndex: int
currLetterIndex: int
nextWordIndex: int
score: int
am: AssetManager
currFirstLetters: Set<Character>
SinglePlayerModel(difficulty, AssetManager, int): void
populateDisplayedList(): void
typedLetter(char): void
getWordsList(difficulty): void
updateWordsDisplayed(): void
getScore(): int
getCurrWord(): String
getCurrWordIndex(): int
getCurrLetterIndex(): int

Fig. 4. Project 1 Iteration 1

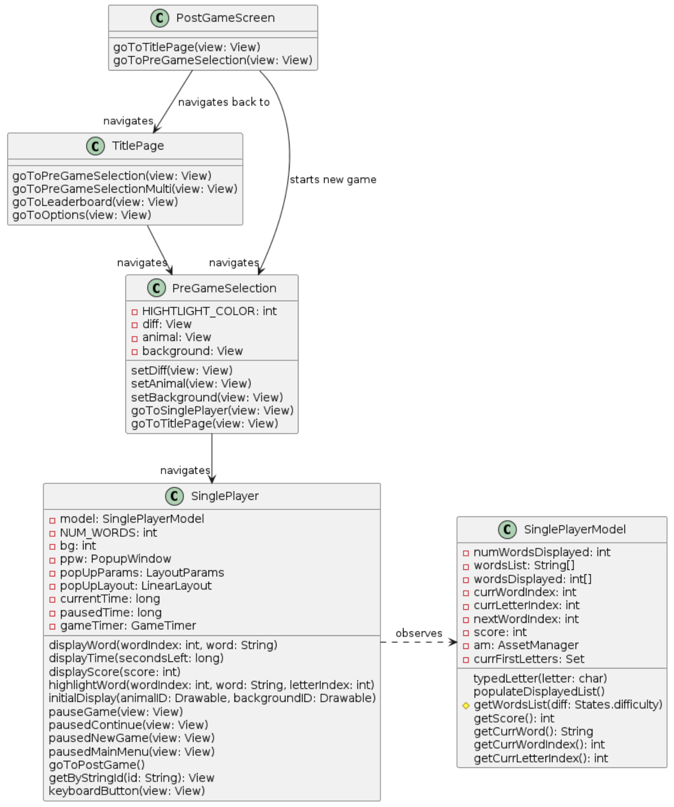
PostGameScreen
goToTitlePage(view: View)
goToPreGameSelection(view: View)
navigates back to
navigates
starts new game
TitlePage
goToPreGameSelection(view: View)
goToPreGameSelectionMulti(view: View)
goToLeaderboard(view: View)
goToOptions(view: View)
navigates
navigates
PreGameSelection
HIGHTLIGHT_COLOR: int
diff: View
animal: View
background: View
setDiff(view: View)
setAnimal(view: View)
setBackground(view: View)
goToSinglePlayer(view: View)
goToTitlePage(view: View)
navigates
SinglePlayer
model: SinglePlayerModel
NUM_WORDS: int
bg: int
ppw: PopupWindow
popUpParams: LayoutParams
popUpLayout: LinearLayout
currentTime: long
pausedTime: long
gameTimer: GameTimer
displayWord(wordIndex: int, word: String)
displayTime(secondsLeft: long)
displayScore(score: int)
highlightWord(wordIndex: int, word: String, letterIndex: int)
initialDisplay(animalID: Drawable, backgroundID: Drawable)
pauseGame(view: View)
pausedContinue(view: View)
pausedNewGame(view: View)
pausedMainMenu(view: View)
goToPostGame()
getByStringId(id: String): View
keyboardButton(view: View)
observes
SinglePlayerModel
numWordsDisplayed: int
wordsList: String[]
wordsDisplayed: int[]
currWordIndex: int
currLetterIndex: int
nextWordIndex: int
score: int
am: AssetManager
currFirstLetters: Set
typedLetter(letter: char)
populateDisplayedList()
getWordsList(diff: States.difficulty)
getScore(): int
getCurrWord(): String
getCurrWordIndex(): int
getCurrLetterIndex(): int

Fig. 5. Project 1 Iteration 2

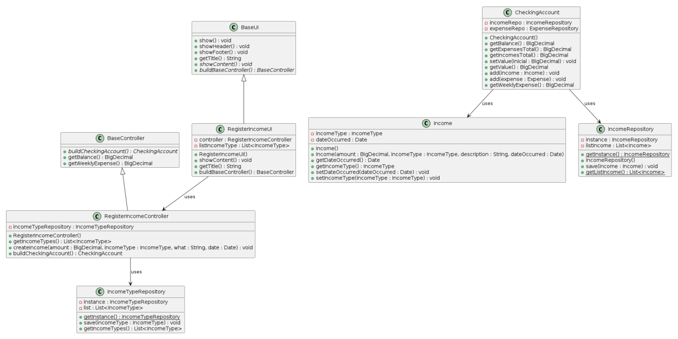
BaseUI
show() : void
showHeader() : void
showFooter() : void
getTitle() : String
showContent() : void
buildBaseController() : BaseController
CheckingAccount
incomeRepo : IncomeRepository
expenseRepo : ExpenseRepository
CheckingAccount()
getBalance() : BigDecimal
getExpensesTotal() : BigDecimal
getIncomesTotal() : BigDecimal
setValue(inicial : BigDecimal) : void
getValue() : BigDecimal
add(income : Income) : void
add(expense : Expense) : void
getWeeklyExpense() : BigDecimal
uses
uses
BaseController
buildCheckingAccount() : CheckingAccount
getBalance() : BigDecimal
getWeeklyExpense() : BigDecimal
RegisterIncomeUI
controller : RegisterIncomeController
listIncomeType : List<IncomeType>
RegisterIncomeUI()
showContent() : void
getTitle() : String
buildBaseController() : BaseController
Income
incomeType : IncomeType
dateOccurred : Date
Income()
Income(amount : BigDecimal, incomeType : IncomeType, description : String, dateOccurred : Date)
getDateOccurred() : Date
getIncomeType() : IncomeType
setDateOccurred(dateOccurred : Date) : void
setIncomeType(incomeType : IncomeType) : void
IncomeRepository
instance : IncomeRepository
listIncome : List<Income>
getInstance() : IncomeRepository
IncomeRepository()
save(income : Income) : void
getListIncome() : List<Income>
uses
RegisterIncomeController
incomeTypeRepository : IncomeTypeRepository
RegisterIncomeController()
getIncomeTypes() : List<IncomeType>
createIncome(amount : BigDecimal, incomeType : IncomeType, what : String, date : Date) : void
buildCheckingAccount() : CheckingAccount
uses
IncomeTypeRepository
instance : IncomeTypeRepository
list : List<IncomeType>
getInstance() : IncomeTypeRepository
save(incomeType : IncomeType) : void
getIncomeTypes() : List<IncomeType>

Fig. 6. Project 3 Iteration 1

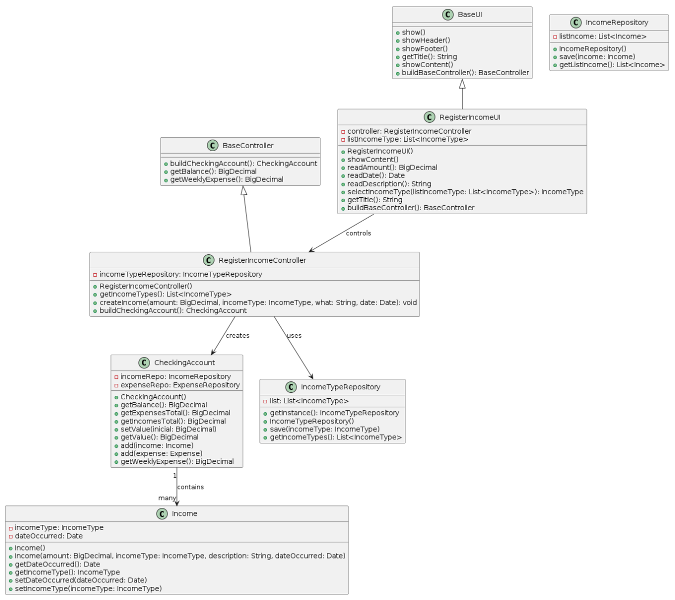
BaseUI
show()
showHeader()
showFooter()
getTitle(): String
showContent()
buildBaseController(): BaseController
IncomeRepository
listIncome: List<Income>
IncomeRepository()
save(income: Income)
getListIncome(): List<Income>
RegisterIncomeUI
controller: RegisterIncomeController
listIncomeType: List<IncomeType>
RegisterIncomeUI()
showContent()
readAmount(): BigDecimal
readDate(): Date
readDescription(): String
selectIncomeType(listIncomeType: List<IncomeType>): IncomeType
getTitle(): String
buildBaseController(): BaseController
BaseController
buildCheckingAccount(): CheckingAccount
getBalance(): BigDecimal
getWeeklyExpense(): BigDecimal
controls
RegisterIncomeController
incomeTypeRepository: IncomeTypeRepository
RegisterIncomeController()
getIncomeTypes(): List<IncomeType>
createIncome(amount: BigDecimal, incomeType: IncomeType, what: String, date: Date): void
buildCheckingAccount(): CheckingAccount
creates
uses
CheckingAccount
incomeRepo: IncomeRepository
expenseRepo: ExpenseRepository
CheckingAccount()
getBalance(): BigDecimal
getExpensesTotal(): BigDecimal
getIncomesTotal(): BigDecimal
setValue(inicial: BigDecimal)
getValue(): BigDecimal
add(income: Income)
add(expense: Expense)
getWeeklyExpense(): BigDecimal
IncomeTypeRepository
list: List<IncomeType>
getInstance(): IncomeTypeRepository
IncomeTypeRepository()
save(incomeType: IncomeType)
getIncomeTypes(): List<IncomeType>
1
contains
many
Income
incomeType: IncomeType
dateOccurred: Date
Income()
Income(amount: BigDecimal, incomeType: IncomeType, description: String, dateOccurred: Date)
getDateOccurred(): Date
getIncomeType(): IncomeType
setDateOccurred(dateOccurred: Date)
setIncomeType(incomeType: IncomeType)

Fig. 7. Project 3 Iteration 2

C. *Class diagram generation*

We are yet to make any progress on this, as this experiment relies heavily on the pre-experiment results. This will be worked on in the coming weeks.

V. Limitations

A. *Internal threats to validity*

To avoid conducting rigorous reviews of code in order to evaluate human-generated diagrams, which as found by Zhang et al. [9] involves prolonged manual effort. It was assumed, for the purposes of this research, that the human generated diagrams were well abstracted and provided the best possible overview for the system at hand given the state of the codebase. Zhang et al. spent a considerable amount of time into determining the specific commit at which the class diagram best matched the codebase, in every project's commit history [9]. However, as taxing as the manual effort was for Zhang et al., there are no guarantees that the class diagram at the chosen commits provide the best possible overview for the system. Nonetheless, the diagrams chosen by Zhang et al. provide a good enough overview of the system so that the findings from the comparison with the GPT-4 generated diagrams are still valuable.

B. *External threats to validity*

Some limitations arise as a consequence of using the same projects as Zhang et al. [9]. Firstly, the selected projects are all java projects, have short periods of activity spanning between 1 and 28 months, have less than 10 contributors each and were all picked from GitHub. A consequence of this, as discussed by Zhang et al. [9] is that the projects may have limited generalizability. This could for example affect the applicability of the results to broader industrial use cases of class diagrams. The tools used in this research, mainly PlantUML, PlantText and GPT-4 are not constrained to any particular programming language or project size. However, further research is required to extend the work in this research to other languages, frameworks etc.

C. *Construct threats to validity*

The Correctness Ratio (CR) and the Mean Rate of Abstraction (MRA) are crucial for the evaluation of the outputted class diagrams. However, these metrics may prove to be insufficient in the process of capturing all relevant differences between the LLM and human-generated diagrams. This may skew the results in either direction, hence neither being an accurate representation of the true differences between the two diagrams nor the actual efficacy of GPT4 for class diagram generation.

The prompt is also an important point to focus on. As mentioned the prompt is essential to generate a quality output from LLMs [10], and our results might not accurately reflect the capabilities of GPT-4 if our chosen prompt is lackluster.

References

[1] M. H. Osman and M. R. Chaudron, "Uml usage in open source software development: A field study." in *EESSMod@ MoDELS*, 2013, pp. 23–32.

[2] R. Koschke, "Architecture reconstruction: Tutorial on reverse engineering to the architectural level," *International Summer School on Software Engineering*, pp. 149–182, 2006.

[3] M. H. Osman, M. R. Chaudron, and P. Van Der Putten, "An analysis of machine learning algorithms for condensing reverse engineered class diagrams," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 140–149.

[4] F. Thung, D. Lo, M. H. Osman, and M. R. Chaudron, "Condensing class diagrams by analyzing design and network metrics using optimistic classification," in *Proceedings of the 22nd International Conference on Program Comprehension*, 2014, pp. 110–121.

[5] X. Yang, D. Lo, X. Xia, and J. Sun, "Condensing class diagrams with minimal manual labeling cost," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 22–31.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[7] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[8] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.

[9] W. Zhang, W. Zhang, D. Strüber, and R. Hebig, "Manual abstraction in the wild: A multiple-case study on oss systems' class diagrams and implementations," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2023, pp. 36–46.

[10] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, "Why johnny can't prompt: how non-ai experts try (and fail) to design llm prompts," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–21.

[11] E. Alpaydin, *Machine learning*. Mit Press, 2021.

[12] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE software*, vol. 7, no. 1, pp. 13–17, 1990.

[13] R. Hebig, T. Ho-Quang, G. Robles, M. A. Fernandez, and M. R. V. Chaudron, "The quest for open source projects that use uml: Mining github," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, Saint-Malo, France, October 2–7 2016, pp. 173–183.

[14] T. Brooks, B. Peebles, C. Homes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. W. Y. Ng, R. Wang, and A. Ramesh, "Video generation models as world simulators," 2024. [Online]. Available: https://openai.com/research/video-generation-models-as-world-simulators

[15] PlantUML, "Plantuml language reference guide," 2024, accessed on 2024-02-20. [Online]. Available: https://plantuml.com/guide

[16] A. Vaughan, PlantUML, Graphviz, A. Editor, J. Sundström, and S. Nichols, "Planttext uml editor," 2024, accessed on 2024-03-07. [Online]. Available: https://www.planttext.com/

[17] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[18] G. Grefenstette, "Tokenization," in *Syntactic wordclass tagging*. Springer, 1999, pp. 117–133.

[19] L. Michelbacher, "Multi-word tokenization for natural language processing," 2013.

[20] W. Depue and OpenAI, "What's new with dall·e-3," 2023, accessed on 2024-02-23. [Online]. Available: https://cookbook.openai.com/articles/what_is_new_with_dalle_3

[21] H. Alkaissi and S. I. McFarlane, "Artificial hallucinations in chatgpt: implications in scientific writing," *Cureus*, vol. 15, no. 2, 2023.

[22] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[23] Vectara and HuggingFace, "Hughes hallucination evaluation model (hhem) leaderboard," 2024, accessed on 2024-02-20. [Online]. Available: https://huggingface.co/spaces/vectara/leaderboard

[24] S. Shahbaz and V. Campanello, "Plantuml comparison script (compare.py)," 2024, accessed on 2024-04-17. [Online]. Available: https://github.com/sh4r10/thesis-autogen-classdiagrams/blob/main/artefacts/scripts/compare.py

[25] ——, "Github repository containing all artifacts," 2024, accessed on 2024-04-18. [Online]. Available: https://github.com/sh4r10/thesis-autogen-classdiagrams/tree/main