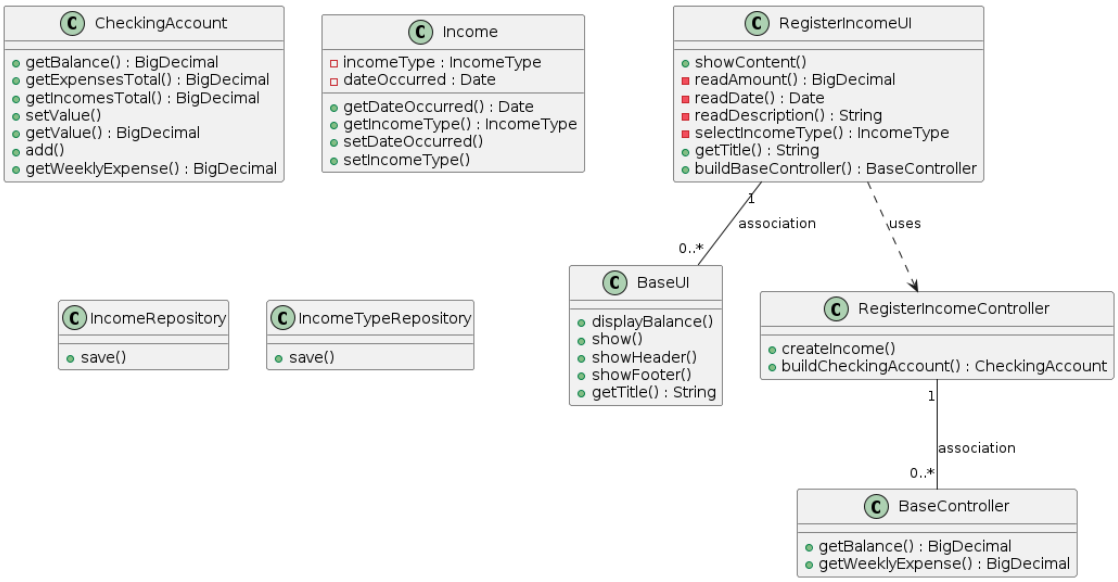
CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



The use of Large Language Models in the Reverse Engineering of Class Diagrams

Bachelor of Science Thesis in Software Engineering and Management

Victor Campanello

Shariq Shahbaz

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Supervisor: Vladislav Indykov, Daniel Strüber
Examiner: Lucas Gren

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: A class diagram created by GPT-4.]

Department of Computer Science and Engineering
University of Gothenburg
Chalmers University of Technology
Gothenburg, Sweden 2024

The use of Large Language Models in the Reverse Engineering of Class Diagrams

Campanello, Victor
University of Gothenburg
guscampvi@student.gu.se

Shahbaz, Shariq
University of Gothenburg
gusshahbsh@student.gu.se

Abstract - Class diagrams are crucial for understanding software projects. They provide an abstract overview of the codebase and aid in onboarding new developers. Traditional reverse engineering tools generate class diagrams directly from code but often produce cluttered diagrams due to their inability to perform abstractions over nonessential elements. This paper explores the potential use of Large Language Models (LLMs), specifically GPT-4, in generating class diagrams from code. We conducted a controlled experiment involving the generation of class diagrams using GPT-4 and compared these diagrams to human-made ones from five Java projects. A prompt was made for each project, both including and excluding the Human Abstraction Framework, which was repeated five times, giving us 50 GPT generated diagrams. The output quality is measured using several metrics namely, *precision*, *recall*, *F1 score* and a *comparison score*.

The inclusion of the Human Abstraction Framework did not show statistically significant improvements in the quality of the diagrams. The study concludes that further research is needed into utilizing GPT-4 for class diagram generation before applying it in a software development context. This is especially true for relationships between classes, where many differences between GPT-4 diagrams and human diagrams were found.

I. Introduction

Class diagrams are an important tool for the understanding of software projects. They provide an abstract overview of projects and can be used as an onboarding 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 diagram's efficacy in helping understand 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 using specific reverse engineering tools that can generate class diagrams from inputted code. The problem is that these tools 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 need for manual labelling 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 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 an LLM to reproduce existing class diagrams by giving it code as input.

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 aims to answer three questions. The first 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. To address the second research question, we will once again generate class diagrams, but this time include the *Human Abstraction Framework* in the prompt to imitate the human abstractions detailed by Zhang et al. [9]. To answer the third research question, we will compare the results of the previous two to see what difference the Human Abstraction Framework made in the diagram generation process. To conclude, this research aims specifically to explore the following questions:

RQ1 How effectively does GPT-4 abstract code relationships when generating class diagrams 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?

The research questions will be answered via a controlled experiment, comparing the performance of GPT-4 in creating class diagrams 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 onboarding of new developers with the assurance that the software models are accurate and up-to-date.

II. Related Work

A. 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 [10]. On the other hand, reverse engineering is the process of creating a representation of an already existing system [11]. 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 future research can utilize to improve the labelling process of reversed-engineered class diagrams.

Although the goals of this research do not involve the labeling of diagrams, the contents of this paper are still 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. To the best of our knowledge, this has not yet been done and hence is the purpose of this research.

B. Human abstractions vs LLMs

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* [12] 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. These three categories are defined in Table I. We will call the cases of real abstractions the *Human Abstraction Framework (HAF)*, see Table II.

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

The need for a categorized framework is vital in establishing what steps need to be followed to make sure the LLM is making the correct decisions when it excludes information from its created class diagrams. This is due to the fact that some information is critical and should not be abstracted away while other information can be removed or simplified to improve the 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.

1) Large Language Models and PlantUML: There is a vast variety of large deep learning models available that we found relevant. Image-generating models, such as DALL-E and Stable Diffusion, output images given text as input. Newer video-generating models, such as Sora, can generate videos using text, but given its multimodal nature, it is also capable of taking in images and videos as input [13].

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].

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.

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 choice [14], 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 [14], which can easily be converted into class diagram images through PlantText [15].

A good way to effectively utilize these large language models is with the technique called *Chain-of-Thought*. Wei et al. [16] accurately describe it as a series of intermediate reasoning steps where that then give you the answer once each step has been performed. Wei et al. [16] likens it to a large math problem that needs several smaller steps to solve. The same can then be applied to a Large Language Model, where you ask it to write out its answer into a series of intermediate steps that lead to the final answer. This form of prompting yields better results than standard prompting, especially for larger and more complex tasks [16] which describes our research well.

2) Custom GPTs: ChatGPT, the online platform used to access GPT-4 allows for the creation of *custom GPTs* [17]. In traditional prompting, the instructions for creating a class diagram need to be provided in every new prompt created with the LLM. Custom GPTs, however, allow us to provide the LLM with unchanging context, allowing repeated prompting without the need to re-provide the context information. This is a great utility in research such as ours, where the underlying context (i.e. creating class diagrams from code in PlantUML) does not change. It is important to note that the context that is not unchanging will still need to be provided when the custom GPT is prompted, e.g. the specific files for the diagram. Custom GPTs serve as the basis of *Data Collection III-C*.

III. RESEARCH METHODOLOGY

This study aims to perform a controlled experiment, following the structure provided by Wohlin et al. [18], to evaluate the efficacy of GPT-4 for class diagram generation using the Human Abstraction Framework extracted from Zhang et al. [9]. See Table II. The experiment aims to determine whether or not the inclusion of the human abstraction framework, can improve the output quality from GPT-4. The output quality is measured using several metrics, including *precision, recall, f1 score* and *comparison score*, a derived metric for determining the overall quality of a diagram; see *Data Collection III-C*. The experiment consists of two treatments, which are diagrams generated using the prompt with the human abstraction framework included and the diagrams generated using the prompt without the human abstraction framework. The variables of the experiment are defined as follows:

- Independent variable: The Prompt (with or without the human abstraction framework)
- Dependent variables: Precision, Recall, F1 score, Comparison Score

The data collection process utilizes the human-generated diagrams for comparison to see how well GPT-4 can recreate each specific diagram. GPT-4 will be provided the code that the diagram represents as input, along with a specific prompt. This is followed by automatic comparisons of the GPT-generated diagrams to the human-generated diagrams, which outputs the aforementioned metrics, see Subsection*Data Collection* III-C.

In order to evaluate RQ3, concerning the differences in the results between RQ1 and RQ2, i.e. the difference in output before and after the inclusion of the human abstraction framework, a statistical analysis is performed. This is done

using the Mann-Whitney U test and the Delaney-Vargha A12 effect size test (see *Data Analysis III-D*).

First, we test whether or not diagrams as a whole tend to have a higher comparison score after the inclusion of the human abstraction framework. The hypotheses are defined below.

$$H_0 : CS_{WITH\ HAF} \leq CS_{WIHOUT\ HAF}$$

$$H_a : CS_{WITH\ HAF} > CS_{WIHOUT\ HAF}$$

This provides an overview of the general trend, however, in order to gather more nuanced insights about the particular categories of differences (see Table IV), the aforementioned tests are performed for each category.

$$Categories = \{Major, Moderate, Minor\}$$

$$H_0 : F1_{c,\text{HAF}} \leq F1_{c,\text{WIHOUT HAF}} \quad \text{for all } c \in Categories$$

$$H_a : F1_{c,\text{HAF}} > F1_{c,\text{WITHOUT HAF}} \quad \text{for all } c \in Categories$$

The justification for the project and LLM selection, as well as a detailed description of the 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. An important note with the human diagrams is that Zhang et al. detected several *disagreements* and *inaccuracies* (see Table I), which are errors in the diagrams that don't represent anything in the source code. These errors should therefore not be replicated, which caused us to redo the human diagrams where we exclude all errors mentioned by Zhang et al. [9] in their replication package.

An important note about these class diagrams is that they differ in some stylistic choices. For example, developers made different decisions on whether to include parameters in methods, with neither of these decisions being inherently more valid than the other. To account for all the stylistic choices we made Table III that outlines all choices made. These stylistic choices are used for the custom GPTs mentioned in Subsection*Collection Setup* (II-B2).

B. Large Language Model Selection

Provided the information from Subsection*Large Language Models and PlantUML* II-B1, 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 [19]. Tokenization is how NLP models process their input [20], 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 [21]. 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 [22]. Within the context of LLMs, a hallucination is defined as "the generated content that is nonsensical or unfaithful to the provided source content" [23]. According to the Hughes Hallucination Evaluation Model, GPT-4 has the lowest hallucination rate, leading to the highest factual consistency [24]. 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 than its competitors. Hence, we determined it to be the best fit for generating abstracted class diagrams.

The outputted PlantUML will be converted to diagram images through PlantText and be analyzed according to the metrics in *Data Analysis (Sect. III-D)*.

C. Data collection

1) Collection Setup: The data collection process uses a chain of two custom GPTs, (see Subsection*Custom GPTs II-B2*). The first custom GPT remains the same for all projects

TABLE III
STYLISTIC CHOICES AND THEIR USAGE

Stylistic choices	Project 1	Project 2	Project 3	Project 4	Project 5
Include getters	no	no	yes	some	no
Include setters	yes	some	yes	some	no
Include methods	some	some	some	some	no
Include constructors	some	no	no	no	no
Include attributes	some	some	some	some	no
Include attribute types	yes	yes	yes	no	no
Include return types	no	yes	yes	no	no
Include access modifiers	yes	no	some	yes	no
Include relationships	yes	some	some	some	some
Include parameter names	yes	some	no	no	no
Include parameter types	yes	yes	yes	no	no
Convert aggregation relationships into association relationships with multiplicities	no	no	yes	yes	no
Convert composition relationships into association relationships with multiplicities	yes	no	yes	no	no

and is responsible for generating a base diagram which includes as much detail in the diagram as possible. This GPT is given the code of the exact class files that exist in the human diagram as attached files and then it generates a PlantUML file. Do note that not all classes of the entire software project are given to GPT, this is due to the fact that the human diagrams only show a small section of each project.

The second custom GPT in the chain, which is different for each project, is responsible for performing abstractions and adhering to the stylistic choices of each project (see Table III).

The second GPT needs to both be different depending on the specific project and whether or not the human abstraction framework is included. This causes us to create 10 Custom GPTs for the second step, two per project, one with the inclusion of HAF and the other one without. An example instruction for the second custom GPT corresponding to project one and project two without HAF is shown in figure 1 and figure 2. All the prompt for all projects (with and without HAF) is included in the replication package [25].

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 our LLM prompts could support the different stylistic choices through configuration options.

2) Raw Data: The chain of custom GPTs is used to generate a class diagram which is then compared to the human diagrams. The comparison is performed through a script, which allows for the repeatable and reliable collection of *raw data* [25]. The raw data consists of 7 metrics, namely: *true positives, false positives, false negatives, precision, recall* and *F1 score*. These metrics are used for answering RQ1 and RQ2 and also for the comparisons in RQ3. The experiment involves performing five trials, for two subgroups (one with the human abstraction framework in the prompt and one without), for each of the projects. This leads to two collections of

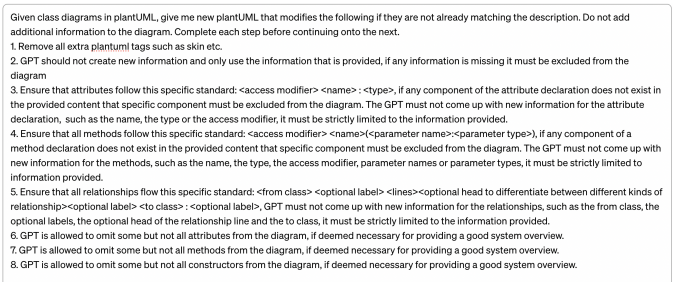

Given class diagrams in plantUML, give me new plantUML that modifies the following if they are not already matching the description. Do not add additional information to the diagram. Complete each step before continuing onto the next.
1. Remove all extra plantuml tags such as skin etc.
2. GPT should not create new information and only use the information that is provided, if any information is missing it must be excluded from the diagram
3. Ensure that attributes follow this specific standard: <access modifier> <name> : <type>, if any component of the attribute declaration does not exist in the provided content that specific component must be excluded from the diagram. The GPT must not come up with new information for the attribute declaration, such as the name, the type or the access modifier, it must be strictly limited to the information provided.
4. Ensure that all methods follow this specific standard: <access modifier> <name>(<parameter name>:<parameter type>), if any component of a method declaration does not exist in the provided content that specific component must be excluded from the diagram. The GPT must not come up with new information for the methods, such as the name, the type, the access modifier, parameter names or parameter types, it must be strictly limited to information provided.
5. Ensure that all relationships flow this specific standard: <from class> <optional label> <lines><optional head to differentiate between different kinds of relationship><optional label> <to class> : <optional label>, GPT must not come up with new information for the relationships, such as the from class, the optional labels, the optional head of the relationship line and the to class, it must be strictly limited to the information provided.
6. GPT is allowed to omit some but not all attributes from the diagram, if deemed necessary for providing a good system overview.
7. GPT is allowed to omit some but not all methods from the diagram, if deemed necessary for providing a good system overview.
8. GPT is allowed to omit some but not all constructors from the diagram, if deemed necessary for providing a good system overview.

Fig. 1. Custom GPT for project 1 with no HAF

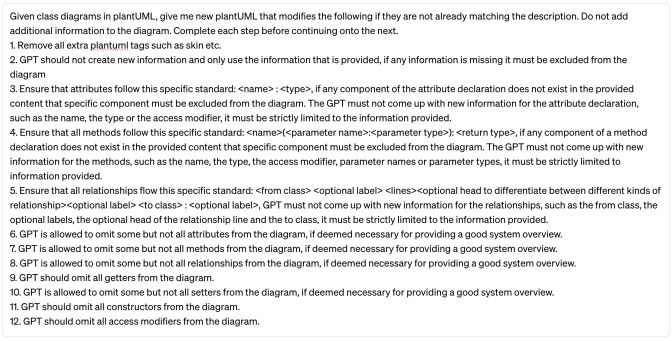

Given class diagrams in plantUML, give me new plantUML that modifies the following if they are not already matching the description. Do not add additional information to the diagram. Complete each step before continuing onto the next.
1. Remove all extra plantuml tags such as skin etc.
2. GPT should not create new information and only use the information that is provided, if any information is missing it must be excluded from the diagram
3. Ensure that attributes follow this specific standard: <name> : <type>, if any component of the attribute declaration does not exist in the provided content that specific component must be excluded from the diagram. The GPT must not come up with new information for the attribute declaration, such as the name, the type or the access modifier, it must be strictly limited to the information provided.
4. Ensure that all methods follow this specific standard: <name>(<parameter name>:<parameter type>): <return type>, if any component of a method declaration does not exist in the provided content that specific component must be excluded from the diagram. The GPT must not come up with new information for the methods, such as the name, the type, the access modifier, parameter names or parameter types, it must be strictly limited to information provided.
5. Ensure that all relationships flow this specific standard: <from class> <optional label> <lines><optional head to differentiate between different kinds of relationship><optional label> <to class> : <optional label>, GPT must not come up with new information for the relationships, such as the from class, the optional labels, the optional head of the relationship line and the to class, it must be strictly limited to the information provided.
6. GPT is allowed to omit some but not all attributes from the diagram, if deemed necessary for providing a good system overview.
7. GPT is allowed to omit some but not all methods from the diagram, if deemed necessary for providing a good system overview.
8. GPT is allowed to omit some but not all relationships from the diagram, if deemed necessary for providing a good system overview.
9. GPT should omit all getters from the diagram.
10. GPT is allowed to omit some but not all setters from the diagram, if deemed necessary for providing a good system overview.
11. GPT should omit all constructors from the diagram.
12. GPT should omit all access modifiers from the diagram.

Fig. 2. Custom GPT for project 2 with no HAF

five diagrams (and their raw data from the comparison) per project. Each collection of raw data corresponding to a particular subgroup of the experiment (e.g all five trial diagrams corresponding to the group without the human abstraction framework gathered for project one), is further processed by another script [25], which is able to derive the mean, standard deviation and variance of the f1 scores for the given collection. This script also determines a *comparison score* for each diagram in the collection, which is used as a metric to determine overall diagram quality.

An important consideration for the comparison between human and GPT diagrams is that not all differences between the diagrams should be valued equally. For example, a difference in the access modifier of an attribute has a smaller impact on the completeness of the diagram as compared to a difference in the inclusion/exclusion of a particular class. This leads us to create a set of categories to sort differences based on their significance level, as shown in Table IV.

TABLE IV
CATEGORIES OF ELEMENT DIFFERENCES

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

The elements in each category are grouped based on their importance to the overall completeness of class diagrams. It is important to note that the *moderate* category is seen as the inclusion of an entire attribute/method, while the *minor* category concerns itself with the details of the attributes/methods(and also the multiplicity of relationships).

The automated comparison script calculates the aforementioned 7 metrics: *true positives, false positives, false negatives, precision, recall* and *F1 score*. These metrics are detailed below.

True positives, false positives and false negatives are used in the calculation of the *precision*, *recall* and *F1 scores* metrics, they are defined as follows.

- *True Positive*: GPT-4 includes an element in the diagram that is included in the human-generated diagram.
- *False Positive*: GPT-4 includes an element in the diagram that is not included in the human-generated diagram.
- *False Negative*: GPT-4 does not include an element in the diagram that is included in the human-generated diagram.

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 cost of false positives is high, i.e. an element 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 element 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:

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

The comparison script [25] is used to compare the two diagrams. The diagrams are compared and the raw data is collected.

Along with calculating the above values for the given files, the script also calculates a comparison score for each of the diagrams in question. The comparison score is a measure used to determine the best diagram from a given collection of diagrams. In order to determine the best diagram, we must first assign weights to the different categories. This is done because of the fact that an error in the Major category is more critical than the Moderate and Minor categories, the same applies for Moderate where Minor is deemed less critical. Hence in order to determine the best diagram, we weigh the categories as shown in *Table V*, where the weight values are based on the deemed importance of the elements in each category.

TABLE V
WEIGHTS OF THE CATEGORIES FOR COMPARISON SCORE CALCULATION

Major	0.6
Moderate	0.3
Minor	0.1

From manual inspection of the data, we determined that the major category tends to have fewer elements than the others. Since the major category is weighed the highest, this can skew the results too much in favour of the diagram with better performance in the major category, while poor performance in the other categories will be ignored. Therefore, the comparison score calculation also takes into account the ratio of elements in a given category to all elements present in the diagram.

$$Ratio_{Category} = \frac{Number\ of\ Elements\ in\ the\ Category}{Number\ of\ Elements\ in\ the\ Diagram}$$

Hence the final comparison score calculation for a given diagram is defined as follows.

$$Categories = \{Major, Moderate, Minor\}$$

$$CS = \sum_{i \in Categories} Ratio_i \cdot Weight_i \cdot F1\ Score_i$$

Performing this for the metrics of *Comparison Score* as well as *F1 Score* for the *Major*, *Moderate* and Minor categories

would lead to four datasets consisting of two independent samples, with five values each. Each value corresponds to a specific diagram of a given project that performed the best in terms of the metric in question out of the five trials.

D. Data analysis

In order to evaluate the hypothesis for RQ3 concerning the two independent samples, we used a Mann-Whitney U-test [26]. This particular test was chosen as the collected data is non-parametric, has one factor and used two treatments [18]. If statistical significance had been found, an A12 effect size test from Vargha and Delaney would have been performed to evaluate the magnitude of the differences between RQ1 and RQ2 [27], however, this did not end up being the case.

The Mann-Whitney U test was used to evaluate the following hypotheses, with a significance level α of 0.05. Each of the 8 independent samples for the four hypothesis tests had a sample size $n = 5$.

$$H_0 : CS_{WITH\ HAF} \leq CS_{WIHOUT\ HAF}$$

$$H_a : CS_{WITH\ HAF} > CS_{WIHOUT\ HAF}$$

$$Categories = \{Major, Moderate, Minor\}$$

$$H_0 : F1_{c,\text{HAF}} \leq F1_{c,\text{WIHOUT HAF}} \quad \text{for all } c \in Categories$$

$$H_a : F1_{c,\text{HAF}} > F1_{c,\text{WITHOUT HAF}} \quad \text{for all } c \in Categories$$

Performing the analysis on a category-by-category basis allowed us to gather more nuanced insights into the effect of the human abstraction framework on the output from GPT-4.

The analysis itself was performed through an automated script [25]. For which, an existing implementation of the Mann-Whitney u-test from *SciPy* provided by Virtanen et al. was used [28] [26]. The outputted p-value from the Mann-Whitney U-test is used to determine the statistical significance of the effect of a particular factor among independent samples. Specifically, if the p-value is less than 0.05, the treatment effect is statistically significant; otherwise, if the p-value is equal to or greater than 0.05, the treatment effect is not statistically significant.

IV. Results

In order to answer RQ1 and RQ2, we selected the best diagram out of the 5 trials. This was done once for each of the following metrics: *Comparison Score* and the *F1 Scores* for the *Major, Moderate and Minor* categories (see Table IV). This results in a collection of 4 datasets, consisting of two groups (with and without HAF) each and 5 samples per group. The resulting dataset for the comparison score is found in Table VI and the datasets for the categories can be found in Table **??**.

TABLE VI
COMPARISON SCORES FOR THE TWO INDEPENDENT SAMPLES

Project	**Without HAF**	**With HAF**
Project 1	14.46	14.28
Project 2	14.30	16.30
Project 3	18.37	16.59
Project 4	18.03	19.16
Project 5	42.86	42.86

TABLE VII
PRECISION, RECALL AND F1 SCORES FOR THE BEST DIAGRAMS IN THE MAJOR CATEGORY

Project	**Precision**	**Recall**	**F1**
Without HAF			
Project 1	0.75	0.86	0.80
Project 2	1.00	0.46	0.63
Project 3	0.67	0.53	0.59
Project 4	1.00	0.8	0.89
Project 5	0.83	0.83	0.83
With HAF			
Project 1	0.83	0.50	0.63
Project 2	1.00	0.45	0.63
Project 3	0.78	0.47	0.58
Project 4	1.00	0.80	0.89
Project 5	0.90	0.90	0.90

TABLE VIII
PRECISION, RECALL AND F1 SCORES FOR THE BEST DIAGRAMS IN THE MODERATE CATEGORY

Project	**Precision**	**Recall**	**F1**
Without HAF			
Project 1	1.00	0.67	0.80
Project 2	0.45	0.73	0.56
Project 3	0.75	0.66	0.70
Project 4	0.65	0.85	0.74
Project 5	0.00	0.00	0.00
With HAF			
Project 1	0.52	0.35	0.42
Project 2	0.60	0.70	0.65
Project 3	0.73	0.63	0.68
Project 4	0.78	0.85	0.81
Project 5	0.00	0.00	0.00

Note that Project 5, does not have any elements in the moderate category hence a value of 0.00 is expected.

TABLE IX
PRECISION, RECALL AND F1 SCORES FOR THE BEST DIAGRAMS IN THE MINOR CATEGORY

Project	Precision	Recall	F1
Without HAF			
Project 1	0.70	1.00	0.83
Project 2	0.76	0.84	0.80
Project 3	0.84	0.97	0.90
Project 4	0.74	0.87	0.80
Project 5	0.55	1.00	0.71
With HAF			
Project 1	1.00	1.00	1.00
Project 2	1.00	1.00	1.00
Project 3	0.96	1.00	0.98
Project 4	0.80	1.00	0.89
Project 5	0.67	1.00	0.80

Table X shows high variance, in terms of the σ standard deviation, for the four datasets along with their μ mean values.

TABLE X
THE MEAN AND THE STANDARD DEVIATION OF THE 4 DATASETS

Dataset	$\mu_{\text{Without HAF}}$	μ_{HAF}	$\sigma_{\text{Without HAF}}$	$\sigma_{\text{With HAF}}$
Comp Score	22.17	22.13	11.68	11.77
F1-Major	0.75	0.73	0.13	0.16
F1-Moderate	0.70	0.64	0.10	0.16
F1-Minor	0.80	0.93	0.07	0.09

For RQ1 (without HAF), Tables VII, VIII and IX show a balance between *precision* and *recall*, with some exceptions depending on the project in question. Although the balance between *precision* and *recall* is similar in RQ2 (with HAF) to RQ1, the tables show that the values for RQ2 tend to be lower than the RQ1, with the exception of the Minor Category (Table IX).

However, to accurately determine the significance of the differences between the results from the first two research questions and to answer RQ3, we performed two statistical tests (see Subsection*Data Analysis III-D*). Table XI, shows the U and p values produced as a result of the Mann-Whitney U.

TABLE XI
RESULTS FROM THE MANN-WHITNEY U TESTS
$\alpha = 0.05, n_1 = n_2 = 5$

Values	Comp. Score	Major	Moderate	Minor
U-value	12.5	14.0	14.0	14.0
p-value	0.54	0.42	0.42	0.42
Median(Without HAF)	18.03	0.80	0.70	0.80
Median(HAF)	16.59	0.63	0.65	0.98

The results from the Mann-Whitney U tests, show no statistical significance, i.e. the p-value for all hypotheses tested is greater than 0.05, hence none of the null-hypotheses can be rejected. In the context of this research, this means that the inclusion of the human abstraction framework within the prompt cannot be said to improve the quality of the gpt-generated diagrams, this is true on the metrics of *Comparison Score*, and the *F1 Scores* for all categories. As the results from the Mann-Whitney U-test did not show any statistical significance, we chose not to perform the Delaney-Vargha A12 effect size test as the test is used to measure the magnitude of the significance.

V. DISCUSSION

A. Experiment findings

From the results of the experiment (see *Results* IV) we make the following findings.

As previously noted, the Mann-Whitney U test shows no statistical significance, however, that does not necessarily mean that there is no statistical significance to be found. A reason for the lack of statistical significance can be that the sample size was too small. This is supported by the high variability in the samples (see standard deviation σ in Table X) [18].

We observed that regardless of whether or not the *human abstraction framework* was included in the prompt, GPT-4 tends to have a balance between the *precision* and *recall* metrics. In the context of this research, this means that GPT-4's ability to include the relevant elements that are indeed relevant (*precision*) is on par with its ability to include most of the actually relevant elements in the diagram (*recall*). The mean F1 score μ across HAF and Without HAF is 0.77, which means that overall GPT-4 is quite good at determining and including the relevant information in the diagram.

A problem that we observed is that GPT-4 is bad at making the same abstractions as the human diagrams. Including more details in the prompt of what should be excluded, could potentially create more accurate abstractions. However, this contradicts the idea of using GPT-4, as being too specific about which classes and elements to exclude could lead to similar amount of effort as creating the diagram manually. GPT performs the best it comes to completely excluding elements, but these cases could be solved by using non-LLM solutions.

We also observed that there is a very low amount of relationships present in all diagrams made by GPT, in the Major Category (with and without HAF). By using a python script [25], we observed that out of all 50 diagrams, there were 466 relationship differences between GPT and Human diagrams, with only 8 similarities. An example of this is shown between Figure 3 and Figure 4. Here GPT-4 does not make the correct inheritance relationship between Upgrade, Food and Item; instead adding associations between Upgrade, Food and ItemVisitor. Figure 5 shows the actual source code where one can clearly see the inheritance.

This shows a huge weakness in GPT's ability to create diagrams, as the relationships between the classes show a lot of information regarding the function of the different classes

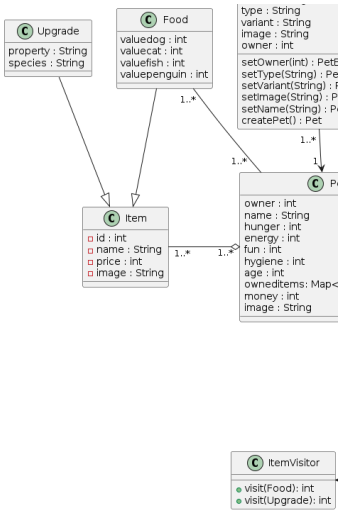


Fig. 3. Human diagram for Project 2

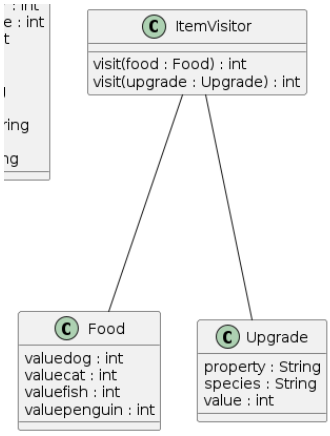


Fig. 4. GPT-4 diagram for project 2, trial 3, HAF included

in a given software project. We speculate that this is because detecting a relationship between two classes is a much harder task compared to attributes and methods, as there are several different ways a relationship manifests itself in source code.

B. Future work

We see the following directions for future work: First, it would be worthwhile to conduct further studies with additional projects in order to either cement or reject our findings. Second, there is also potential to expand our work into other similar areas. Our research focused specifically on class diagrams of Java projects using GPT-4 but it is not restricted to this criteria. The research can be expanded upon through the inclusion of other programming languages that utilize class diagrams, such as Kotlin, C#, Python etc. The research can also be performed with other potentially better and newer LLMs in the future, to see whether or not significant improvements have been made (such as more accurate relationship mappings). The prompting techniques and insights can be used to expand the work to diagrams other than class diagrams, such as but not

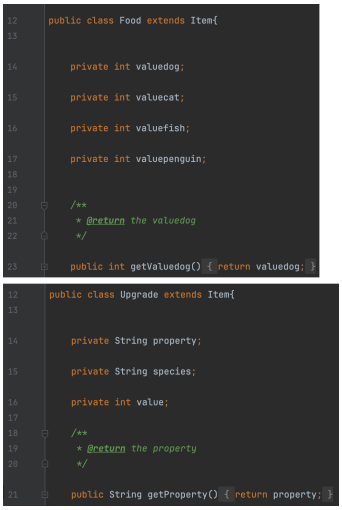


Fig. 5. Images showing sections of source code showing inheritance

limited to sequence, component and deployment diagrams. Using different languages and LLMs could be done using the same scripts we have created [25], but diagrams other than class diagrams would require additional work to evaluate the correctness of the generated diagrams. Third, it would be of interest to include the source code in the comparison. This would allow you to be more detailed in discerning the differences between different kinds of false positives and get a better understanding of the causes of different false positives. This would be insightful, as some differences may be caused by hallucinations from GPT with no connection to the source code while others are elements that exist in the source code but were abstracted way by the human diagram creators. From a cursory glance and based on our intuit knowledge of the source code we observed a low amount of hallucinations but this should be studied further to better and more accurately understand the source of the error. Fourth, we analysed the different class diagrams and made a list of stylistic choices, these can be used to create a tool that dynamically creates a prompt and interfaces with an LLM to generate class diagrams that meet the specific needs of the user.

VI. Threats to validity

A. Internal threats to validity

To avoid conducting rigorous reviews of source code in order to evaluate human-generated diagrams, which as found by Zhang et al. [9] involves prolonged manual effort. Therefore, for this research we decided to use diagrams from the original project creators, for the purpose of recreating them with GPT-4. However, a problem that may arise from this decision is that GPT-4 may be significantly worse at performing the specific abstractions performed by the humans, but not bad at performing abstractions overall.

Additionally, 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 chosen commits provide the best match between the diagram and source code. Nonetheless, the diagrams chosen by Zhang et al. provide a good enough overview of the system, so the findings from the comparison with the GPT-4 generated diagrams are still valuable. This is evident from the results, where the *F1 score* across HAF and Without HAF has a mean value μ of 0.77.

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 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, see *Future Work* (Subsection V-B).

C. Construct threats to validity

The Comparison Score is an important metric for measuring overall diagram quality, however, the calculation for it is dependent on arbitrary weight assignments. These weights may skew the results in either direction, hence the metric might neither being an accurate representation of the true differences between the two diagrams nor the actual efficacy of GPT4 for class diagram generation. In order to mitigate this risk, we perform a statistical analysis on a category-by-category basis using the *F1 score*, which allows us to get more nuanced insights into each category while at the same time confirming the observations from the comparison score analysis.

The prompt is also an important point to focus on. As mentioned, the prompt is essential to generate quality output from LLMs [29], and our results might not accurately reflect the capabilities of GPT-4 if our chosen prompt is lacklustre. In order to mitigate this risk, we spent a considerable amount of testing and fine-tuning the prompts in an iterative manner [25]. We followed the relevant literature to implement modern prompt engineering techniques, such as chain-of-thought (see Subsection *Large Language Models and PlantUML* II-B1) [16].

VII. Conclusion

In this paper, we used GPT-4 to generate class diagrams for five Java software projects, comparing each GPT-generated diagram to the diagram made by original the project developers. This process involved the formulation of project-specific prompts, using techniques such as chain-of-thought, based on the stylistic choices of the human-generated diagram. We used two treatments, which consisted of the inclusion and exclusion of the human abstraction framework from the prompt. These prompts were used to create 5 diagrams per treatment for each of the projects, in order to account for variance, resulting in the creation of 50 diagrams. From the created diagrams, we picked the best diagrams on the metrics of *comparison score*, and *F1 scores* for the *major*, *moderate* and *minor* categories of differences, creating four datasets of two independent samples with 5 values each. After performing the hypothesis tests, we determined the inclusion of the human abstraction framework to be statistically insignificant for improving GPT-generated diagram output quality. This, along with other metrics collected throughout the research, leads us to conclude that further work is required in order for large language models to be suitable for use in a software development context.

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

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

[12] 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.

[13] 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

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

[15] 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/

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[17] G. Tao, S. Cheng, Z. Zhang, J. Zhu, G. Shen, and X. Zhang, "Opening a pandora's box: Things you should know in the era of custom gpts," *arXiv preprint arXiv:2401.00905*, 2023.

[18] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

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

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

[21] 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

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

[23] 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.

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

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

[26] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[27] 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.

[28] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[29] 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.