JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

On the Use of GPT-4 in the Reverse Engineering of Class Diagrams

Victor Campanello[†,‡], **Shariq Shahbaz**[†,‡], **Vladislav Indykov**[†,‡], **and Daniel Strüber**[†,‡,*]
[†]University of Gothenburg, Sweden
[‡]Chalmers University of Technology, Sweden
[*]Radboud University, Netherlands

ABSTRACT *Class diagrams* are a standard notation for effectively visualizing the structure of a software system in the context of software design and analysis. In particular, class diagrams are widely used in *reverse engineering*, the main goal of which is to reconstruct and analyze the design of a system from a given codebase to understand and improve it. Yet, traditional reverse engineering tools that generate class diagrams from code often produce cluttered outputs due to their inability to perform abstraction, that is, leaving out or summarizing nonessential elements in a way human experts would do.

In this paper, we explore the use of large language models, specifically GPT-4, in generating class diagrams from code to emulate human abstraction. We used an experimental methodology in which we applied GPT-4 to a dataset of five substantial projects, comprising 4452 code elements and their expert-created abstraction to 338 model elements. Our prompts were informed by an in-depth manual analysis of the dataset, in which we identified stylistic choices that can lead to different generation outcomes and, therefore, are useful to include as hints into the prompt to reflect user preferences. To understand GPT-4's inherent ability to abstract, we experimented with including hints from the Human Abstraction Framework (HAF), a previous systematization of human abstraction, into the prompts. Our results shed a promising light on the use of GPT-4 for making abstraction decisions at a fine level of granularity (e.g., the inclusion of attribute- and operation-level and type information), where mean F1 scores of 91% and 89% could be achieved, respectively, while more coarse-grained abstraction decisions (especially regarding the representation of relationships) lead to considerably worse F1 scores between 62% and 75%. The inclusion of HAF-based hints into prompts did not significantly affect accuracy, shedding a promising light on GPT-4's inherent abstraction ability. Our results emphasize the need for further research on understanding the handling of relationships during manual abstraction.

KEYWORDS Class Diagrams, Reverse Engineering, Large Language Models

1. Introduction

Class diagrams are an essential tool for the understanding of software projects. They provide an abstract overview of the system structure and can be used as an onboarding tool for developers and maintainers. However, a substantial problem arises due to the low frequency at which the diagrams are updated (Osman & Chaudron 2013). This may cause a roadblock since the efficacy of diagrams relies on the relevance and accuracy of the codebase representation, and regular manual updates of class diagrams are laborious and error-prone. A potential response to this dilemma can be using reverse engineering tools that can generate class diagrams from code (Siala et al. 2024). However, the main problem with existing tools is that they cannot perform abstractions (Koschke 2006), which results in diagrams that look cluttered.

When creating class diagrams, human experts perform ab-

JOT reference format:
Victor Campanello, Shariq Shahbaz, Vladislav Indykov, and Daniel Strüber. *On the Use of GPT-4 in the Reverse Engineering of Class Diagrams.* Journal of Object Technology. Vol. 24, No. 2, 2025. http://dx.doi.org/10.5381/jot.2025.24.2.a14

straction by making decisions about whether to include certain information in the diagram or not, and if so, how to include it. Zhang et al. (Zhang et al. 2023) empirically studied manual abstraction by categorizing abstraction decisions at the level of classes, relationships, attributes, and operations. Their study could inform the development of reverse engineering tools that mimic human abstraction.

A first step in this direction was made with the previous approaches by Osman et al. (Osman et al. 2013) and Thung et al. (Thung et al. 2014), who both used traditional machine learning techniques to condense reverse-engineered class diagrams. They focused on the decision of whether to include or exclude entire classes from the class diagram, which represents a coarse-grained understanding of abstraction. However, as suggested by Zhang et al. (Zhang et al. 2023), the abstractions performed by developers in practice are more nuanced, and represent a spectrum from *coarse-grained* (such as including entire classes) to *fine-grained* (such as including specific methods, operations, and parameters). In addition, while both techniques were effective in predicting relevant classes, they required a substantial overhead for users in labeling class diagrams for the same system to obtain training data.

Hypothetically, the use of Large Language Models (LLM) could contribute to addressing these issues. There has been a significant increase in the capabilities of LLMs over the last few years (Brown et al. 2020), which have shown to have an edge over traditional machine learning techniques for many tasks. 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 for software development and documentation. In the context of reverse engineering, the focus should be on how they handle large codebases as input, as well as their ability to understand the various complex relationships in the code. Therefore, the purpose of this paper is to analyze, organize, and fine-tune the output from an LLM to reproduce existing class diagrams by giving it code as input.

In this study, we focus on the LLM GPT-4 because at the time of writing it is one of the most accurate LLMs available (Achiam et al. 2023),(Minaee et al. 2024). GPT-4 is a multi-modal LLM that outperforms a variety of other LLMs in hallucination and common-sense reasoning metrics (Minaee et al. 2024).

We address and answer three research questions:

RQ1: How does GPT-4 perform when generating class diagrams using only the code as input?
RQ2: How does GPT-4 perform when generating class diagrams, using *Human Abstraction Framework* as input along with code?
RQ3: How does the inclusion of the *Human Abstraction Framework* within the prompt affect the outputted diagrams in comparison with diagrams created without this framework?

RQ1 and RQ2 both put GPT-4's abstraction capabilities to test, as we use it to generate class diagrams from code, and compare the result to human-created diagrams. The difference between both questions is that in RQ2, by explicitly including information from the *Human Abstraction Framework* (Zhang et al. 2023) in the prompts, we can study the impact of explicitly guiding GPT-4 with information about the expected abstractions. HAF provides a collection of abstraction cases on the level of classes, attributes, operations, and relationships, describing how humans typically abstract when they create class diagrams. In RQ3, we compare the results of the previous two questions to see what difference the Human Abstraction Framework made in the diagram generation process.

We answered the research questions 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. For answering RQ1 and RQ2, GPT-4's performance is measured using the metrics of *precision*, *recall*, *F1-score* (described later in detail) across three categories of differences (see Table 5).

For answering RQ3, we perform statistical analysis for the *F1-score* metrics, quantifying the differences in accuracy metrics obtained from RQ1 and R2, of using only code, and both code and HAF prompt hints as input, respectively. The answers to these questions will act as a stepping stone for future research to use these rapidly improving LLMs to streamline the software documentation and the reverse engineering processes.

2. Related Work and Background

Reverse Engineering of Class Diagrams. Reverse engineering is the process of creating a representation of an already existing system (Chikofsky & Cross 1990). In the context of model-driven reverse engineering, software models, especially those from UML and domain-specific model languages, are used as the representation of choice (Hughes & Bae 2024; Priefer et al. 2021; Peldszus et al. 2018; Vaupel et al. 2015). A particularly widely used representation in this context, which is used by practical reverse engineering tools such as MoDisco (Bruneliere et al. 2014), are class diagrams. Yet, the automatic creation of class diagrams from code naturally leads to a cluttered and hard-to-read representation if it represents the same information as the code itself.

One of the ways to deal with excessive information in automatically generated diagrams is to use machine learning. Machine learning is broadly defined as the process of using past data to create an algorithm to predict future data (Alpaydin 2021). A line of work has used machine learning to address the problem of *condensing* reverse-engineered class diagrams by removing irrelevant information. Osman et al. (Osman et al. 2013) were the first to apply a collection of traditional machine learning techniques to label classes as either relevant or irrelevant, based on statistical metrics such as the number of attributes and operations, and available labels that were manually created by developers for the same project. Thung et al. (Thung et al. 2014) improved on this initial work by considering additional, network-based metrics, while also reducing the effort for manual labeling by automatically assigning labels to a subset of unlabelled data. Follow-up work by Yang et al. (Yang et al. 2016) uses a different combination of machine learning techniques, in particular, unsupervised and ensemble ones, to improve the accuracy of the classification.

However, all of these existing approaches focus on the coarse

Table 1 Definitions of key terms, from Zhang et al. (Zhang et al. 2023)

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.

abstraction task of excluding or including entire classes, which does not capture the full range of abstraction decisions done by human experts (see below). In addition, while they are able to achieve a good accuracy, they all have a remaining effort for initial manual labeling, which is generally laborious. In this work, we focus on an approach that can automatically make abstraction decisions at different levels of granularity, and avoids the manual labeling effort of traditional machine learning techniques.

Human Abstractions in Class Diagrams. We based our understanding of human abstraction on the empirical study by Zhang et al. (Zhang et al. 2023), who investigated how developers create abstractions from code when they create class diagrams. Zhang et al. manually collected a set of five substantial software projects with class diagrams from the *Lindholmen Dataset* (Hebig et al. 2016) 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 (Zhang et al. 2023). Zhang et al. broadly categorized differences between the source code and the diagram into a taxonomy of *real abstractions*, *disagreements,* and *inaccuracies*. These three categories are defined in Table 1. The same authors later (in parallel to the present work) validated their taxonomy based on four additional subject projects (Zhang et al. 2025), where it demonstrated generalizability to larger cases.

From these differences, our focus is on real abstraction. Zhang et al. identified a number of *cases* of real abstraction, which we call *Human Abstraction Framework (HAF)* in this paper. We show the definition of these cases in Table 2. Importantly, abstractions performed in class diagrams go well beyond the exclusion or inclusion of whole classes, which is a formative insight for automated approaches to mimic human abstraction.

AI support for class diagram generation. A line of recent work has analyzed the capability of large language models to create UML-like models from various data sources. Considering class diagram generation, which is the focus of our work, considered sources include requirements descriptions (Wang et al. 2024; Ferrari et al. 2024), domain descriptions (Chen et al. 2023; Cámara et al. 2023), and images (Conrardy & Cabot 2024). Other related work in this direction focuses on UML use case enhancement in an interactive, feedback-driven process (De Vito et al. 2023), and generation of data flow diagrams from user stories (Herwanto 2024).

Table 2 Cases of Real Abstraction, from (Zhang et al. 2023)

Classes	
Inheritance Structure Omission	Inheritance structure(s) from the source code are not presented in the model.
Class Omission	Class(es) from the source code of the modeled system part are not presented in the model.
Class Summary	Two or more classes from the source code are presented as one class in the model.
Attributes	
Attribute Omission	Attribute(s) from the source code are not presented in the model.
Attribute Summary	Multiple attributes from the source code are presented as one attribute in the model.
Attribute Type Omission	The type of attribute(s) from the source code is not presented in the model.
Default Value Omission	Attribute(s) from the source code have a default value not presented in the model.
Operations	
Operation Omission	Operation(s) from the source code are not presented in the model.
Operation Summary	Multiple operations from the source code are presented as one operation in the model.
Parameter Omission	Parameter(s) from the source code are not presented in the model.
Parameter Name Omission	Parameter name(s) from the source code are not presented in the model.
Return Type Omission	The return type of method(s) from the source code is not presented in the model.
Collection Type Underspecification	The types of object(s) that can be stored in collections are not presented in the model, or only the types of object(s) are presented without showing the collection.
Relationships	
Relationship Omission	Relationship(s) from the source code are not presented in the model.
Relationship Loosening	Attribute(s) (i.e., owned elements) from the source code are modeled as named associations (and not as compositions or aggregations).
Relationship Summary	For two classes that access each other's values indirectly via a third class in the source code, a direct association is presented in the model.

Compared to this line of previous work, our study is the first to consider the creation of class diagrams from *code*, in the context of reverse engineering. The specific technical challenges that arise in this context are the identification of useful cases for training, the prompt engineering (specifically, deciding how to present the code information to the LLM), as well as the performance evaluation of the resulting LLM application in this particular task.

Large Language Models and PlantUML. There is a vast variety of large language models available that we found relevant

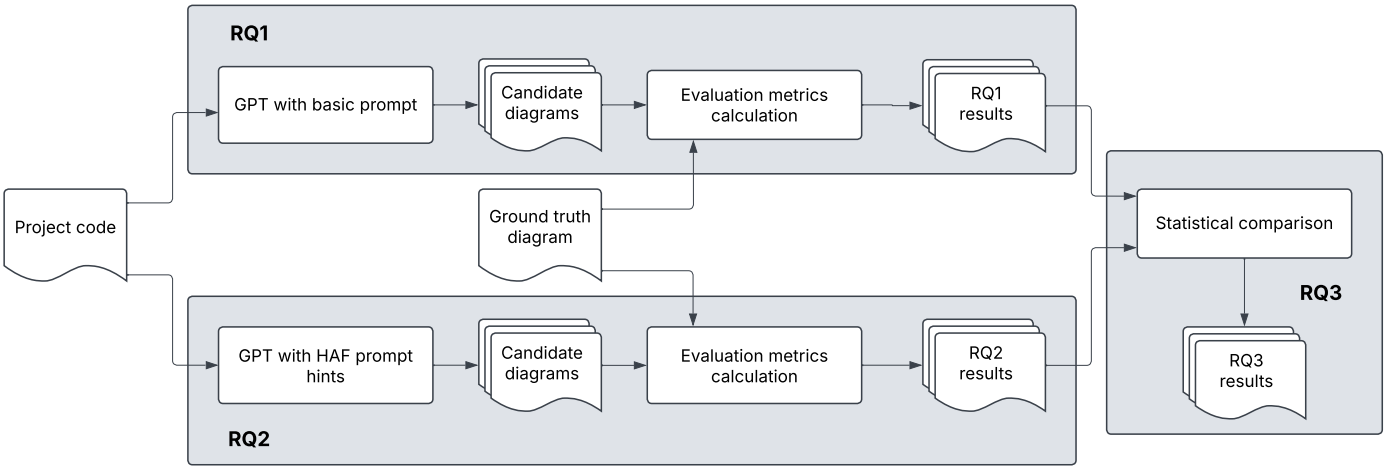

Figure 1 Overview of experimental methodology

to consider for our methodology. 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 (Brooks et al. 2024). 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 (Achiam et al. 2023). However, as these models are limited to text as their output. Hence, we require a textual representations for class diagrams, ideally one that also allows for easy graphical visualization. Our solution of choice is PlantUML (Vaughan & Sundström 2024), which fulfills all of these requirements: it 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 (Vaughan & Sundström 2024), which can easily be converted into class diagram images through PlantText (Vaughan et al. 2024).

A good way to effectively utilize these large language models is with the technique called *Chain-of-Thought*. Wei et al. (Wei et al. 2022) accurately describe it as a series of intermediate reasoning steps that give the answer once each step has been performed. Wei et al. (Wei et al. 2022) liken 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 (Wei et al. 2022) which describes our research well.

Custom GPTs. ChatGPT, the online platform used to access GPT-4 allows for the creation of *custom GPTs* (Tao et al. 2023). 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 a permanent 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 still needs to be provided when the custom GPT is prompted, e.g., the specific files for the diagram.

3. Methodology

Recall that our goal is threefold: to assess GPT'4 abstraction ability when provided with just code (RQ1), with both code and hints from Zhang et al.'s Human Abstraction Framework (RQ2), and to evaluate the difference between these two approaches (RQ3). To address this goal, we performed a controlled experiment, following the methodological guidlines provided by Wohlin et al. (Wohlin et al. 2012). The experiment, illustrated in the overview in Fig. 1, consists of two treatments, the results of which provide answers to RQ1 and RQ2: diagrams generated using the prompt with and without the human abstraction framework included. Our *independent variable* is the prompt used for the experiment (with or without the human abstraction framework). Our *dependent variables* are evaluation metrics of precision, recall, and F1-score. To answer RQ3, we performed a statistical comparison by applying hypothesis testing to the results from RQ1 and RQ2.

The data collection process utilized human-generated diagrams for comparison to see how well GPT-4 can recreate each specific diagram. GPT-4 was provided with 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 evaluation metrics. 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 standard null hypothesis significance testing.

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

3.1. 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 projects can then be given to GPT-4 as input and their resulting class diagrams can be compared to the existing one created by humans previously.

This research utilized the same five projects as Zhang et al. (Zhang et al. 2023), summarized in Table 3. All projects use Java as their programming language, a language interesting to consider in combination with class diagrams, since it includes concepts of classes, fields, and methods, which are also part of class diagrams, as its key structures. 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. Zhang et al. manually matched 466 classes, 1352 attributes, and 2634 operations from source code to 338 model elements (Zhang et al. 2023). 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 1). To avoid that errors bias our evaluation of abstraction ability, we recreated the human diagrams in such way that all errors mentioned by Zhang et al. (Zhang et al. 2023) in their replication package were removed.

Importantly, our considered class diagrams differed in several stylistic choices. For example, developers made different decisions about whether to include parameters in methods or not, and it is impossible to claim if one decision was more valid than another. An evaluation of an automatic technique for class diagram extraction should not penalize the use of a different style preference by the technique compared to a human expert. To this end, we systematically considered all diagrams to extract a collection of relevant stylistic choices made, outlined in Table 4.

Table 3 Projects with information on code and model elements: classes, operations, attributes

ID	Name	Code elements			Model elements		
		cl.	op.	att.	cl.	op.	att.
1	ZooTypers	15	77	52	6	31	16
2	RaiseMeUp	40	545	474	17	59	43
3	EAPLI PL 2NB	60	255	51	8	30	9
4	FreeDaysIntern	92	502	216	8	35	15
5	NeurophChanges	259	1255	559	10	0	0

Projects on GitHub: ZooTypers, RaiseMeUp, EAPLI_PL_2NB, FreeDaysIntern, NeurophChanges

A further concern is that human abstraction does not necessarily have a single, unique correct outcome - different developers might create different class diagrams for the same underlying code base. To address the impact of developer variety, we repeated the LLM-based class diagram process 5 times per system and considered the best-obtained class diagram for our reporting of evaluation metrics.

Table 4 Stylistic choices and their usage within the projects

Stylistic choices	Proj. 1	Proj. 2	Proj. 3	Proj. 4	Proj. 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 into association	no	no	yes	yes	no
Convert composition into association	yes	no	yes	no	no

3.2. Large Language Model Selection

Provided the information from subsection *Large Language Models and PlantUML* 2, we decided that Image and text-generating models are the most relevant for the goals of this research due to the type of their outputs. 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 (Grefenstette 1999). Tokenization is how NLP models process their input (Michelbacher 2013), but each model is limited by the number of tokens it can process, which limits how large 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 (Depue 2023). 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 (Minaee et al. 2024). 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 (Alkaissi & McFarlane 2023). Within the context of LLMs, a hallucination is defined as "the generated content that is nonsensical or unfaithful to the provided source content"(Ji et al. 2023). According to the Hughes Hallucination Evaluation Model, GPT-4 has the lowest hallucination rate, leading to the highest factual consistency (Hughes & Bae 2024). 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 (Minaee et al. 2024). A dataset consisting of 70,000 multichoice questions called HellaSwag ranked GPT-4 the highest at 95.3 percent correct, 6.44 points above second place (Minaee et al. 2024).

Provided the above reasoning in combination with the results from Minaee et al.(Minaee et al. 2024), we decided to use GPT-4 for our research. This is because GPT-4 is well-balanced in its reasoning and hallucinates far less often than its competitors. Hence, we determined it to be the best fit for generating abstracted class diagrams.

3.3. Diagram generation.

The diagram generation process uses a chain of two custom GPTs, (see *Custom GPTs* in Sect. 2). The first custom GPT remains the same for all projects and is responsible for generating a base diagram based on the provided code in PlantUML format, including as much detail in the diagram as possible. 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 4) and based only on the *PlantUML* generated by the first custom GPT in the chain. We isolated the second GPT from the source code to focus it entirely on abstraction and stylistic adaptation without consideration of the specific details from the source.

The second GPT needs to accommodate both the specific project and whether the human abstraction framework is included or not (RQ1 vs. RQ2). Consequently, we created 10 custom GPTs for the second step, two per project, one with the inclusion of HAF and the other without. An example of an LLM prompt is presented in Figure 2, while full prompts for all the projects (with and without HAF) are included in the Supplementary Artifact (Campanello et al. 2024). Each of the prompts starts with the same overall goal description and the same first five steps. Steps 1–2 establish a baseline of what the diagram should look like, whereas steps 3–5 aim to remove potential variance and errors. The following steps encode stylistic preferences that are specific to each project, and could generally be generated from user preferences and general style settings and themes, based on an automated tool (discussed in Section 5.1).

Based on the cases of real abstraction from Table 2, we developed hints for including the *Human Abstraction Framework (HAF)*, which were added before any other prompts in the corresponding experiments. The structure of our HAF-based prompt hints is presented in Figure 3. This prompt cites the relevant definitions from the HAF verbatim. Notably, this part of the prompt is the same for each used project, whereas the basic prompt (indicated with a reference in the last line) contains variable parts reflecting user preferences—see above.

The code presented as input to the first custom GPT comprises Java class files, provided as attached files, for all classes that have a counterpart in the human-created baseline diagram. Providing just a selection, instead of all code files of each project, is a reasonable assumption for our approach: a class diagram visualization can generally be useful for any scope of a larger system, but users are typically understanding in a specific scope, which we, in the case of our subject projects, identify via the classes of the human-created diagram file. Nevertheless, this setup still leaves a considerable design space of abstraction decisions, from coarse ones such as including particular classes and relationships or not, to fine ones, such as the specific information included for class-level elements (explained below), which forms the basis of our evaluation.

3.4. Evaluation metrics (RQ1 and RQ2)

The chain of custom GPTs was used to generate class diagrams which were then compared to the available baseline diagrams created by human experts. For the comparison, we considered the presence and absence of elements in the two class diagrams, based on their names. Based on this information, we calculated seven metrics in total for the comparison, namely: *true positives, false positives, false negatives, precision, recall* and *F1-score*. These metrics are used for answering RQ1 and RQ2, and serve as input 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 five diagrams (and their evaluation metric result data) per project. The metrics calculation was performed through a script that is publicly available to support repeatable and reliable data collection (Campanello et al. 2024). Each collection of metrics 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), was further processed by another script (Campanello et al. 2024) for deriving the mean, standard deviation and variance of the F1-scores for the given collection.

An important consideration for the comparison between human and GPT diagrams is that not all differences between the diagrams should be considered as equal. For example, a difference in the access modifier of an attribute affects the abstraction quality of a diagram in a different way compared to a difference in the inclusion/exclusion of a particular class. This leads us to introduce categories to distinguish differences based on their *granularity*, as shown in Table 5.

The elements in each category are grouped based on their granularity level. It is important to note that the *medium* category is seen as the inclusion of an entire attribute/method, while the *fine* category concerns itself with the details of the attributes/methods.

The automated comparison script calculates the aforemen-

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 the information provided.
5. Ensure that all relationships follow 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 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.

Figure 2 Custom GPT prompt with no HAF (steps 6–12 are specific to project 2)

Table 5 Granularity categories of element differences

Coarse	Classes, Relationships.
Medium	Entire Attributes, Entire Methods.
Fine	Access Modifiers, Attribute Type, Method Return Type, Parameter Name, Parameter Type.

tioned 7 metrics: *true positives, false positives, false negatives, precision, recall* and *F1-score*. These metrics are detailed below.

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

Precision, recall, and F1 score are metrics commonly used to evaluate the performance of classification models, particularly in imbalanced datasets, using the formulaes shown below. They provide insights into the model's ability to correctly classify positive instances. Precision allows us to get a measure of how many of the determined positives are actual positives. Recall allows us to get a measure of how many of the actual positives are determined positive. Precision is a useful measure when the cost of false positives is high. On the other hand, recall is a good measure when the cost of false negatives is high. We have determined that the effort to correct false negatives and false positives of a class diagram is similar, hence we need to find a balance between the precision and recall scores. We use an F1-score to achieve this balance. The F1 score is the harmonic mean of precision and recall.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

To strengthen the robustness of our findings, we performed

GPT is allowed to and should, within reason, try to make the following abstractions over the code in the generated diagram when the instructions specify it:
[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.
Given class diagrams in plantUML, give me new plantUML ***[basic prompt]***

Figure 3 Custom GPT prompt with HAF hints, based on HAF's description from (Zhang et al. 2023)

a sanity check[1], where we, for one our projects, compared the F1 scores when only the first GPT is used, to using the chain of the two GPTs. As expected, we noted a particularly poor abstraction ability when using just the first GPT, which, as it does not discard any elements, produces many false positives (e.g., method return type and attribute type receiving a F1 score of 0%), which increases to 100% after the second GPT is used).

3.5. Statistical comparison (RQ3)

In RQ3, to support a statistical comparison of the results obtained with and without including the HAF into prompt hints (from RQ1 and RQ2, respectively), we conduct hypothesis testing. Considering the *Coarse*, *Medium* and *Fine* categories leads to three 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.

[1] Comparison table for sanity check:
http://htmlpreview.github.io/?https://raw.githubusercontent.com/sh4r10/thesis-autogen-classdiagrams/refs/heads/revision/artefacts/controlled-experiment/sanity-check/comparison.html
Underlying files:
https://github.com/sh4r10/thesis-autogen-classdiagrams/tree/revision/artefacts/controlled-experiment/sanity-check

In order to evaluate the hypothesis for RQ3 concerning the two independent samples, we used a Mann-Whitney U-test (Mann & Whitney 1947). This particular test was chosen as the collected data is non-parametric, has one factor, and uses two treatments (Wohlin et al. 2012). In particular, it is equipped to deal with small sample sizes (Sijtsma & Emons 2010). We evaluated the three pairs of null hypotheses and alternative hypotheses that address the three granularity levels *Coarse*, *Medium*, *Fine* as per Table 5. We used a standard significance threshold α of 0.05. Each of the 10 independent samples for the hypothesis tests had a sample size `n = 5`. Rejecting each null hypothesis would provide support for accepting for the corresponding alternative hypothesis.

$$H_{0_{coarse}} : F1_{coarse,\text{with HAF}} \leq F1_{coarse,\text{without HAF}} \quad (1)$$
$$H_{A_{coarse}} : F1_{coarse,\text{with HAF}} > F1_{coarse,\text{without HAF}} \quad (2)$$
$$H_{0_{medium}} : F1_{medium,\text{with HAF}} \leq F1_{medium,\text{without HAF}} \quad (3)$$
$$H_{A_{medium}} : F1_{medium,\text{with HAF}} > F1_{medium,\text{without HAF}} \quad (4)$$
$$H_{0_{fine}} : F1_{fine,\text{with HAF}} \leq F1_{fine,\text{without HAF}} \quad (5)$$
$$H_{A_{fine}} : F1_{fine,\text{with HAF}} > F1_{fine,\text{without HAF}} \quad (6)$$

Table 6 Overview of precision, recall, and F1-scores based on granularities

	Fine			*Medium*			*Coarse*		
Project	**Precis.**	**Recall**	**F1**	**Precis.**	**Recall**	**F1**	**Precis.**	**Recall**	**F1**
Without HAF									
Project 1	**1.00**	**1.00**	**1.00**	**0.75**	**0.95**	**0.84**	**0.50**	**0.86**	**0.63**
Project 2	**0.68**	**0.79**	**0.73**	0.56	0.65	0.60	**1.00**	**0.50**	**0.67**
Project 3	**0.83**	**1.00**	**0.91**	**0.66**	**0.93**	**0.77**	**0.67**	0.67	**0.67**
Project 4	**1.00**	**1.00**	**1.00**	**0.48**	**0.91**	**0.62**	**1.00**	**0.89**	**0.94**
Project 5	N/A	N/A	N/A	N/A	N/A	N/A	**0.91**	**0.77**	**0.83**
With HAF									
Project 1	0.98	0.98	0.98	0.52	0.59	0.55	**0.50**	0.45	0.48
Project 2	0.66	0.72	0.69	**0.62**	**0.86**	**0.72**	0.50	0.47	0.49
Project 3	0.80	**1.00**	0.89	0.62	0.78	0.69	0.54	**0.70**	0.61
Project 4	**1.00**	**1.00**	**1.00**	**0.48**	**0.91**	**0.62**	**1.00**	0.80	0.89
Project 5	N/A	N/A	N/A	N/A	N/A	N/A	0.59	0.67	0.62

N/A indicates that a metric computation was not possible.

4. Results

To answer RQ1 and RQ2, we selected the best diagram out of the 5 trials, thus accommodating for possible variety in human-created abstractions (described above). This was done once for each of the *F1-scores* for the *Coarse, Medium, and Fine* categories. This results in a collection of 3 datasets, consisting of two groups (with and without HAF) each and 5 samples per group. An overview of the results for these datasets can be found in Table 6. We note that Project 5 does not have any elements in the medium or fine category hence a value cannot be calculated. Table 7 shows statistical summary information for the three datasets, namely, the mean values (μ) and the standard deviations (σ) when the F1-score, as the metric that balances the other two metrics of precision and recall, is considered.

4.1. Accuracy without HAF (RQ1)

Considering the abstraction ability of GPT-4 when not including hints from the HAF in the prompt, Table 6 shows a balance between *precision* and *recall*, with some exceptions depending on the project in question. The mean μ F1-score values across the coarse, medium, and fine categories for RQ1 (Without HAF) are 0.75, 0.70, and 0.91 respectively, see Table 7. To then answer RQ1, it means that not only GPT-4 (on its own) is effective at identifying the relevant elements (precision) and including most of them in the resulting diagram (recall), but it maintains this balance between precision and recall across the three categories of differences. Hence we can state that based on the results of the experiment when generating class diagrams on its own from the source code, GPT-4 performs well across all categories when measured on the metrics of *precision, recall* and *F1-score*.

To give a more detailed picture of these results, Table 8 provides an overview of the results of the types of the elements that are captured by the categories of *coarse*, *medium*, and *fine*, grouped in the table using horizontal lines.

Considering the *coarse* category, remarkably, in all projects

Table 7 The mean and the standard deviation for the F1 scores at 3 granularity levels

Dataset	$\mu_{\textbf{without HAF}}$	$\sigma_{\textbf{without HAF}}$	$\mu_{\textbf{with HAF}}$	$\sigma_{\textbf{with HAF}}$
F1-Coarse	0.75	0.12	0.62	0.15
F1-Medium	0.70	0.10	0.65	0.07
F1-Fine	0.91	0.11	0.89	0.12

and for all considered metrics, we find high values for classes and low values for relationships. For the case of classes, these results can be explained with our preselection of classes that we fed as input to GPT-4 in order to specify the context for the expected class diagrams. Deviations from perfect scores, such as project 2 with 91% and project 3 with 75% for recall, indicate an attempt by GPT-4 to remove classes deemed as irrelevant in that context. Conversely, for relationships, we observe low numbers, between F1 scores of 40% and 0%. We generally observed a low number of relationships present in all diagrams made by GPT, in the Coarse Category (with and without HAF). By using a python script (Campanello et al. 2024), 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 in Figure 4. Here GPT-4 does not create the correct inheritance relationship between Upgrade, Food, and Item; instead adding associations between Upgrade, Food, and ItemVisitor. Figure 5 shows the corresponding source code fed as input to GPT-4 that contains the inheritance.

Another insightful example is Project 1 where GPT-4 is asked to include all relationships and to convert two composition relationships to association relationships. In this instance, even traditional reverse engineering techniques that do not rely on any machine learning algorithms (e.g., IntelliJ class diagram generator without the work of Osman et al., Thung et al., or Yang et al.) would perform better than GPT-4, as GPT-4 very likely fails with all relationships but the traditional reverse engineering would only fail at abstracting the composition relationships. This shows a 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 in a given software project. We speculate that this is because detecting a relationship between two classes is a harder task compared to attributes and methods, as there are several different ways a relationship manifests itself in source code.

In the *medium* category, we find an F1 score of 70% overall. Looking at the individual element types, we see that, compared to classes and elements on the coarse level, there is more variation among specific types, achieving values between 84% and 55% F1 score for attributes, and between 70% and 80% for methods. These overall results suggest a lackluster ability of GPT-4 to emulate human abstraction for medium-granularity elements.

Finally, in the *fine* category, the overall score is particularly high, at 91%. Notably, this finding arises based on only four out

Table 8 Precision, Recall, and F1-scores for the best diagrams for prompts without HAF (RQ1) and with HAF (RQ2)

	Project 1			Project 2			Project 3			Project 4			Project 5		
Elements	**Precis.**	**Recall**	**F1**	**Precis.**	**Recall**	**F1**	**Precis.**	**Recall**	**F1**	**Precis.**	**Recall**	**F1**	**Precis.**	**Recall**	**F1**
Without HAF															
Classes	**1.00**	**1.00**	**1.00**	**1.00**	**0.91**	**0.95**	**1.00**	**0.75**	**0.86**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Relationships	0.00	0.00	0.00	**1.00**	**0.15**	**0.27**	**0.33**	**0.50**	**0.40**	0.00	0.00	0.00	0.00	0.00	0.00
Attributes	**0.73**	**1.00**	**0.84**	**0.71**	0.83	0.77	**0.55**	**0.75**	**0.63**	**0.38**	**1.00**	0.55	N/A	N/A	N/A
Methods	**1.00**	**0.66**	**0.80**	0.51	0.58	**0.70**	**0.70**	**1.00**	**0.83**	**0.63**	**0.83**	0.71	N/A	N/A	N/A
Access modifiers	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.68	**1.00**	0.81	**1.00**	**1.00**	**1.00**	N/A	N/A	N/A
Attribute type	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	N/A	N/A	N/A
Method return type	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.92**	**1.00**	**0.96**	**1.00**	**1.00**	**1.00**	N/A	N/A	N/A
Parameter name	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	N/A	N/A	N/A
Parameter type	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	N/A	N/A	N/A
With HAF															
Classes	**1.00**	0.83	0.91	0.82	0.82	0.82	**1.00**	**0.75**	**0.86**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Relationships	0.00	0.00	0.00	0.00	0.00	0.00	0.14	**0.50**	0.22	0.00	0.00	0.00	0.00	0.00	0.00
Attributes	0.53	0.67	0.59	0.66	**1.00**	**0.79**	**0.55**	**0.75**	**0.63**	**0.38**	**1.00**	**0.56**	N/A	N/A	N/A
Methods	0.52	0.56	0.54	**0.57**	**0.71**	0.63	0.65	0.79	0.71	**0.63**	**0.83**	**0.73**	N/A	N/A	N/A
Access modifiers	**1.00**	**1.00**	**1.00**	**1.00**	0.98	0.99	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	N/A	N/A	N/A
Attribute type	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	N/A	N/A	N/A
Method return type	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.92**	**1.00**	**0.96**	**1.00**	**1.00**	**1.00**	N/A	N/A	N/A
Parameter name	**1.00**	**1.00**	**1.00**	**0.11**	**0.25**	**0.15**	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	N/A	N/A	N/A
Parameter type	0.93	0.93	0.93	**0.08**	**0.08**	**0.08**	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	N/A	N/A	N/A

N/A indicates that a metric computation was not possible.

of the five projects, those that include fine-granularity elements. In the case of project 1, in which all stylistic choices have a value of either yes or no, we observe perfect accuracy, recall and F1, which indicates that GPT remained faithful to the specification. In the projects that had *some*-entries for stylistic choices, we still observe examples of high F1 scores, such as *method return type* in project 3 at 96% F1.

Considering the N/A entries in Table 8, note that the metric computation is not possible especially in Project 5, where the baseline diagram does not contain attributes or methods, rendering it infeasible for us to be able to count how many attributes or methods GPT has correctly removed. Considering instead, for example, F1 or access modifiers for project 2, we observe that the baseline diagram for Project 2 does include attributes; therefore, we are able to count exactly for how many attributes GPT has correctly removed the access modifiers, leading to the entry 1.00.

4.2. Accuracy with HAF (RQ2)

When including hints from the HAF into the prompt, the results obtained, presented in the lower half of Table 6, show a similar balance between *precision* and *recall*, as for RQ1. The mean μ values across the coarse, medium, and fine categories for RQ2 are 0.62, 0.65, and 0.89. To answer RQ2, given the balance between precision and recall, we can state that GPT-4 is equally effective at identifying relevant elements (precision) as it is at including most of the relevant elements in the resulting diagram (recall). Furthermore, on the metric of F1-score, it performs well in the fine ($F1 : 0.89$) category, slightly worse in the coarse ($F1 : 0.62$) and medium ($F1 : 0.65$) categories. Hence, we can state that based on the results of the experiment when generating a class diagram using the human abstraction framework along with the source code, GPT-4 performs very well in the fine and mediocre in the *medium* and *coarse* categories, when measured on the metrics of *precision*, *recall* and *F1-score*.

Although the balance between *precision* and *recall* is similar in RQ2 to RQ1, the table shows that the values for RQ2 tend to be marginally lower than for RQ1. This is supported by the mean values discussed above. Furthermore, the median values for the F1-scores across the categories of differences for the samples without HAF, are higher than the values for the samples with HAF,

Considering the fine category, we observe that for cases where a *some*-entry was provided as part of the style preferences, including HAF lead to drastically better results in cases such as parameter name for project 3, where perfect accuracy, recall and F1 could now be achieved.

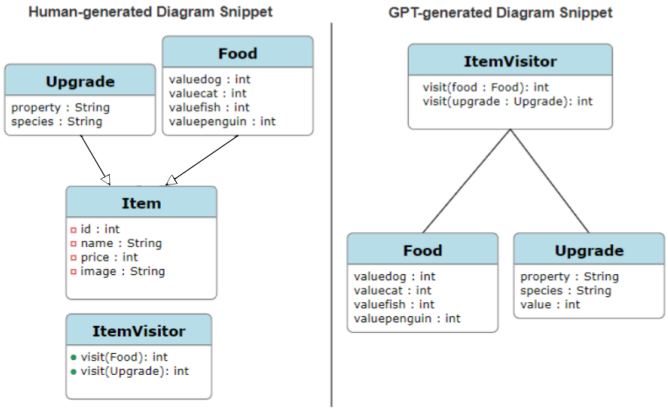

Figure 4 Snippets from human-created vs. GPT-4-created class diagrams (trial 3, HAF included) for Project 2

```java
public class Food extends Item{
    private int valuedog;
    private int valuecat;
    private int valuefish;
    private int valuepenguin;

/* @return the valuedog */

public int getValuedog() {
    return valuedog;}
```

```java
public class Upgrade extends Item{
    private String property;
    private String species;
    private int value;

/* @return the property */

public String getProperty() {
    return property; }
```

Figure 5 Snippets of source code with inheritance

4.3. Comparison (RQ3)

To accurately determine the significance of differences between the results from the RQ1 and RQ2 and to answer RQ3, we performed a Mann-Whitney U-test. The results from the Mann-Whitney U-tests, showed no statistical significance across the different categories of fine, medium and coarse, i.e. the p-value for all hypotheses tested is greater than 0.05. Specifically, we observe $p = 0.97$ for the coarse, $p = 0.81$ for the medium category $p = 0.77$ and for the fine category.

Hence, we cannot reject the null hypotheses for any of the fine, medium or coarse categories. 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 effective at determining and including the relevant information in the diagram.

5. Discussion

We now discuss the implications of our results for tool developers, our contribution to theory building on automation of abstraction capabilities, and threats to validity.

Table 9 The mean and the standard deviation for the F1 scores for all element types.

Dataset	μ_{without} HAF	σ_{without} HAF	μ_{with} HAF	σ_{with} HAF
Classes	0.96	0.05	0.92	0.07
Relationships	0.13	0.17	0.04	0.09
Attributes	0.70	0.11	0.64	0.09
Methods	0.76	0.06	0.65	0.07
Access modifiers	0.95	0.08	1.00	0.00
Attribute type	1.00	0.00	1.00	0.00
Method return type	0.97	0.03	0.99	0.02
Parameter name	0.5	0.5	0.54	0.47
Parameter type	0.5	0.5	0.50	0.46

5.1. Implications for Tool Developers

Overall, our results shed a promising light on the ability of LLMs such as GPT-4 to mimic human abstraction. This is informative for tool developers who want to incorporate LLM-based abstraction capabilities such as those studied in this paper into a reverse engineering tool. Especially in the category of fine-grained abstraction decisions, which were not addressed by previous machine-learning-based approaches (e.g., (Osman et al. 2013) (Thung et al. 2014) (Yang et al. 2016)), and where we observed average F1 scores of 89% and 91%, respectively, we can maintain a new state-of-the-art based on our study. For more coarse-grained changes, where we observe lower F1 scores, we have reported detailed observations regarding potential roadblocks that should be taken into account. For example, since the number of included relationships was generally lower than in human-created diagrams, one can experiment with more explicit hints about the expected numbers and types of represented relationships in the prompt.

While our scope deliberately excluded aspects of UI and user process, it is important to mention that we do not conceive the user to manually interact with the LLM, including manual editing of prompts—an effective reverse engineering tool based on our findings would interact with the LLM "under the hood". Such a tool could benefit from our list of stylistic differences that we created as part of our data collection, based on our manual analysis of the underlying dataset (see Table 4). To customize the output class diagrams to the user's needs, the user could be presented with a list of stylistic choices from which they set the options in the same way we did to recreate the baseline diagrams, to the possible alternatives of *no*, *yes*, and *some*. Sects of typical choices could be bundled into presets, further reducing the manual user involvement. As a form of *model-driven prompt engineering* (Clarisó & Cabot 2023), the tool would automatically generate a suitable prompt based on the choices (comparable to the one in Fig. 2), which is then used

to create the diagram.

Such a tool could balance some of the identified weaknesses of LLMs by complementing them with other methods, for example, using GPT-4 with the HAF prompt included for the fine-grained abstraction decisions, where GPT-4 perfoms particularly well, and a different technique with complementary strengths for coarse- and medium-grained ones.

5.2. Implications for Theory Building.

Strengths and weaknesses. In RQ1 and RQ2, we found that the abstraction capability of GPT-4 is vastly different for different element types. We described in detail the case of relationships, in which the performance of GPT-4 in emulating was particularly low, including the syntactic level – correctly representing specific relationships from the code using appropriate relationship types – as well as the semantic level – correctly distinguishing between relevant and irrelevant relationships, where the latter can be discarded during abstraction. At lower granularity levels, the obtained results are generally better. This finding highlights a gap in our understanding of what relationships designers include and not include when creating class diagrams, which also seems hard to address with large language models.

To mitigate the issues with abstraction of relationships, we need to improve our empirical understanding of what makes relationships relevant. Such insights could be obtained from dedicated studies, both artifact studies and user studies, focusing on questions of which relationships are retained and discarded in practice. Potentially, mathematical techniques from fields such as information retrieval and machine learning can be useful to assign importance weights to references based on their names. Insights from such studies could be included as hints in the prompt, an idea that our findings prove useful for the case of fine-grained abstractions.

Absence of significant impact of including HAF. In RQ3, we found that including hints regarding a conceptual model of human abstraction did not significantly affect GPT-4's ability to create human-like abstractions. There are different possible implications of this finding to theory building:

The lack of a significant effect could be interpreted in two ways: as a negative result for this particular prompt engineering idea, or as a positive result for the inherent abstraction ability of GPT-4. Towards the former, when considering the setup of including abstraction hints into the prompt as a form of prompt engineering, the experiments did not demonstrate the effectiveness of this approach. Towards the latter, one could also see this outcome as a positive result demonstrating that GPT-4 has ingrained knowledge about abstraction, which then makes the explicit provision of this knowledge superfluous. This knowledge could either be a reflection of the training data, or a phenomenon of emergence. Investigating these potential implications further, based on more comprehensive datasets – including the additional subject projects from (Zhang et al. 2025) –, is an exciting area for future research.

5.3. Threats to Validity.

External validity. External validity is threatened by our project selection that affects how our results generalize to other cases. Our selected projects were obtained from an available dataset obtained from the Lindholmen dataset (Zhang et al. 2023). This dataset is generally diverse, as projects do not fulfill many requirements beyond containing a class diagram. In consequence, one strength is that the considered projects span a variety of different application domain. Yet, one restriction is that the project needs to be available from GitHub. It remains open to which extent results obtained on GitHub repositories can generalize to industrial settings. Related threats are that the projects had relatively short periods of activity spanning between 1 and 28 months, and had less than 10 contributors each.

All considered projects use Java as their programming language. The results may generalize to projects in other languages that have the same object-oriented concepts as class diagrams and Java, including C++ and C#. For other programming languages that use entirely different paradigms, we cannot claim generalizability. The abstraction problem for these languages seems entirely different, as class diagrams might not be the notation of choice in these cases.

A potential source of bias is that the projects used in our study also informed the development of the human abstraction framework (Zhang et al. 2023), a consequence of the lack of larger datasets with more alternative projects to choose from. GPT's abstraction ability might be lower than observed when considering projects that use entirely different abstractions than our subject ones. Still, we argue that such projects might be rare, as the majority of considered abstraction cases occurred in multiple of the 5 projects stemming from different domains, which indicates a reasonable level of generalizability to other cases.

The prompts used in our experiments are custom-tailored to the project at hand, based on their stylistic choices (e.g., whether getters, setters and constructors should be included or not). This could be seen as a threat to their generality: for a new project, the prompt needs to be modified. However, our effort to systematically identify such stylistic choices allows to automatically generated such prompts from user preferences (see Section 5.1). In consequence, we are actually reducing the impact of bias, namely, the bias from arbitrary stylistic choices, as compared to actual abstraction ability.

Internal validity. We provide a preselection of classes to ChatGPT, which, arguably, could lead to an overestimation of precision for the *coarse* category. However, given a large system with many classes, one can focus on different scopes of the system to create a class diagram. Without setting a fixed scope, as we do, a mismatch between GPT's results and the manual baseline might not indicate a lack of precision, but a focusing on a different scope. Hence, we argue that our pre-filtering is indeed a necessary step to ensure a fair comparison. Our insights at coarse granularity are not at all positive and highlight a weakness in dealing with references, where we do observe a poor abstraction ability, which, we think, is noteworthy and could be insightful for follow-up research.

A threat to our evaluation metrics is that precision, recall, and F1-score can only be calculated for a given combination of granularity and model if the model calculates elements on that granularity. For the fine and medium granularities, means

that our overall results only stem from 4 of the total of 5 class models, since project 5 lacks relevant elements.

Furthermore, our results might not accurately reflect GPT-4 capabilities if our chosen prompt is lackluster, as good prompts are essential to generate quality output from LLMs (Zamfirescu-Pereira et al. 2023). To mitigate this risk, we spent a considerable amount of testing and iteratively fine-tuning the prompts (Campanello et al. 2024). We followed best practices from the relevant literature, such as chain-of-thought (Wei et al. 2022). One concern with our prompts could be that we mix imperative and third-person styles. From our experience, this mixing of styles is unproblematic in GPT-4, which displays a good ability to abstract from prompt styles. To mitigate this threat, we performed a small additional experiment in which we compared the output of our used prompt to a revised prompt that uses a single style[2]. We found that this change did not significantly change our results.

Construct validity. We operationalize the ability to emulate human abstraction by relying on a comparison to a *ground truth* of a specific human-created abstraction. However, as there might be variation how different experts create abstractions for the same system, there is a risk that we measure the ability to emulate the specific developer who created the abstraction, instead of general abstraction ability. To mitigate this risk, we executed each abstraction task five times and considered the output that was most similar to the ground truth abstraction.

6. 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 the original 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 project, to account for variance, resulting in the creation of 50 diagrams. From the created diagrams, we picked the best diagrams on the *F1-scores* for the *coarse*, *medium*, and *fine* 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. However, we can still observe that GPT-4 is relatively efficient at determining and including the relevant information in the diagrams, but less efficient at making abstractions and building relationships.

We see the following directions for future work: First, it would be worthwhile to conduct further studies with additional projects to either cement or reject our findings. Second, there is also potential to expand our work into other similar areas. Our scope focused specifically on class diagrams of Java projects using GPT-4, but it could be expanded in the future. Importantly, the technical sphere of LLMs is fast-moving, and it is conceivable that alternative, newer approaches might improve accuracy and address some of the issues (such as poor ability to abstract references) reported in this paper. A comprehensive comparative evaluation of several LLMs, including Claude-Sonnet and DeepSearch, is an important direction for future work. The research can be expanded upon through the inclusion of other programming languages that utilize class diagrams, such as Kotlin, C#, Python, etc. The prompting techniques and insights can be used to expand the work to diagrams other than class ones, such as sequence, component, and deployment diagrams. Using different languages and LLMs could be done using our scripts (Campanello et al. 2024). Third, it would be of interest to include the source code in the comparison. This would allow us 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 code but were abstracted away by the human diagram creators. Fourth, to enable a fully automated approach for class diagram reverse engineering, we intend to develop a user-oriented tool that can automate the generation of prompts like those shown in this paper, based on user preferences and by integrating with other tools that work particularly well for other abstractoin decisions than those where GPT-4 excels.

[2] Comparison table for prompt style experiment:
http://htmlpreview.github.io/?https://raw.githubusercontent.com/sh4r10/thesis-autogen-classdiagrams/refs/heads/revision/artefacts/controlled-experiment/sanity-check/comparison.html
Underlying files:
https://github.com/sh4r10/thesis-autogen-classdiagrams/tree/revision/artefacts/prompt-style-experiment

References

Achiam, J., Adler, S., & Agarwal, S. (2023). *GPT-4 technical report.* Open.ai.

Alkaissi, H., & McFarlane, S. I. (2023). Artificial hallucinations in ChatGPT: Implications in scientific writing. *Cureus*, *15*(2), 13–15.

Alpaydin, E. (2021). *Machine learning*. MIT press.

Brooks, T., Peebles, B., & Homes, C. (2024). *Video generation models as world simulators.* Open.ai.

Brown, T., Mann, B., & Ryder, N. (2020). Language models are few-shot learners. In *NeurIPS* (Vol. 33, pp. 1877–1901). Curran Associates, Inc.

Bruneliere, H., Cabot, J., Dupé, G., & Madiot, F. (2014). Modisco: A model driven reverse engineering framework. *IST*, *56*(8), 1012–1032.

Campanello, V., Shahbaz, S., Indykov, V., & Strüber, D. (2024). *Supplementary artifact for 'on the use of GPT-4 in the reverse engineering of class diagrams'.* https://figshare.com/s/6c01826d12037ada78da. Figshare.

Chen, K., Yang, Y., Chen, B., López, J. A. H., Mussbacher, G., & Varró, D. (2023). Automated domain modeling with large language models: A comparative study. In *MODELS* (Vol. 26, pp. 162–172). IEEE.

Chikofsky, E. J., & Cross, J. H. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software*, *7*(1), 13–17.

Clarisó, R., & Cabot, J. (2023). Model-driven prompt engineering. In *MODELS* (Vol. 26, pp. 47–54). IEEE.
Conrardy, A., & Cabot, J. (2024). From image to UML: First results of image-based UML diagram generation using LLMs. *arXiv preprint arXiv:2404.11376*.
Cámara, J., Troya, J., Burgueño, L., & Vallecillo, A. (2023). On the assessment of generative AI in modeling tasks: An experience report with ChatGPT and UML. *SSM*, *22*(3), 781–793.
Depue, W. (2023). *What's new with DALL·E-3.* DALL·E-3.
De Vito, G., Palomba, F., Gravino, C., Di Martino, S., & Ferrucci, F. (2023). Echo: An approach to enhance use case quality exploiting large language models. In *SEAA* (Vol. 49, pp. 53–60). IEEE.
Ferrari, A., Abualhaija, S., & Arora, C. (2024). Model generation from requirements with LLMs: An exploratory study. *arXiv preprint arXiv:2404.06371*.
Grefenstette, G. (1999). Tokenization. In *Syntactic wordclass tagging* (Vol. 9, pp. 117–133). Springer.
Hebig, R., Ho-Quang, T., Robles, G., Fernandez, M. A., & Chaudron, M. R. V. (2016). The quest for open source projects that use UML: Mining github. In *MODELS* (Vol. 3, pp. 1–11). ACM.
Herwanto, G. B. (2024). Automating data flow diagram generation from user stories using large language models. In *NLP4RE* (Vol. 7). HAL.
Hughes, S., & Bae, M. (2024). *Hughes hallucination evaluation model leaderboard.* Huggingface.
Ji, Z., Lee, N., Frieske, R., Yu, T., & Su, D. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, *55*(12), 1–38.
Koschke, R. (2006). Architecture reconstruction: Tutorial on reverse engineering to the architectural level. *SE*, *5413*, 140–173.
Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, *18*(1), 50–60.
Michelbacher, L. (2013). *Multi-word tokenization for natural language processing* (PhD Thesis). University of Stuttgart.
Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large language models: A survey. *arXiv preprint:2402.06196*.
Osman, M. H., & Chaudron, M. R. (2013). UML usage in open source software development: A field study. In *MODELS* (Vol. 1, pp. 23–32). ACM.
Osman, M. H., Chaudron, M. R., & Van Der Putten, P. (2013). An analysis of machine learning algorithms for condensing reverse engineered class diagrams. In *ICSM* (Vol. 26, pp. 140–149). IEEE.
Peldszus, S., Strüber, D., & Jürjens, J. (2018). Model-based security analysis of feature-oriented software product lines. In *Proceedings of the 17th acm sigplan international conference on generative programming: Concepts and experiences* (pp. 93–106).
Priefer, D., Rost, W., Strüber, D., Taentzer, G., & Kneisel, P. (2021). Applying mdd in the content management system domain: Scenarios, tooling, and a mixed-method empirical assessment. *Software and Systems Modeling*, *20*, 1919–1943.
Siala, H. A., Lano, K., & Alfraihi, H. (2024). Model-driven approaches for reverse engineering-a systematic literature review. *IEEE Access*, *12*, 62558–62580.
Sijtsma, K., & Emons, W. (2010). Nonparametric statistical methods. *International encyclopedia of education*, 347–353.
Tao, G., Cheng, S., Wu, Z., Zhu, Y., et al. (2023). Opening a pandora's box: Things you should know in the era of custom GPTs. *arXiv preprint arXiv:2401.00905*.
Thung, F., Lo, D., Osman, M. H., & Chaudron, M. R. (2014). Condensing class diagrams by analyzing design and network metrics using optimistic classification. In *ICPC* (Vol. 22, pp. 110–121). ACM.
Vaughan, A., & Sundström, J. (2024). *PlantUML language reference guide.* PlantText UML.
Vaughan, A., Sundström, J., & Nichols, S. (2024). *PlantText UML editor.* PlantText UML.
Vaupel, S., Strüber, D., Rieger, F., & Taentzer, G. (2015). Agile bottom-up development of domain-specific ides for model-driven development. In *Flexmde@ models* (pp. 12–21).
Wang, B., Wang, C., Liang, P., Li, B., & Zeng, C. (2024). How LLMs aid in UML modeling: An exploratory study with novice analysts. *arXiv preprint arXiv:2404.17739*.
Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., … Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS* (Vol. 35, pp. 24824–24837). Curran Associates, Inc.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering* (Vol. 236). Springer.
Yang, X., Lo, D., Xia, X., & Sun, J. (2016). Condensing class diagrams with minimal manual labeling cost. In *COMPSAC* (Vol. 40, pp. 22–31). IEEE.
Zamfirescu-Pereira, J., Wong, R. Y., Hartmann, B., & Yang, Q. (2023). Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *CHI* (Vol. 1, pp. 1–21). ACM.
Zhang, W., Zhang, W., Strüber, D., & Hebig, R. (2023). Manual abstraction in the wild: A multiple-case study on OSS systems' class diagrams and implementations. In *MODELS* (Vol. 26, pp. 36–46). IEEE.
Zhang, W., Zhang, W., Strüber, D., & Hebig, R. (2025). An empirical study of manual abstraction between class diagrams and code of open source systems. *SoSyM'25: Software & Systems Modeling*. (to appear)

About the authors

Victor Campanello and Shariq Shahbaz have recently completed their bachelor studies in the Software Engineering and Management program of Gothenburg University.

Vladislav Indykov is a PhD student at Chalmers and Gothenburg University. His website: https://euphort.se/

Daniel Strüber is an associate professor at Chalmers and Gothenburg University, and an assistant professor at Radboud University. His website: https://www.danielstrueber.de