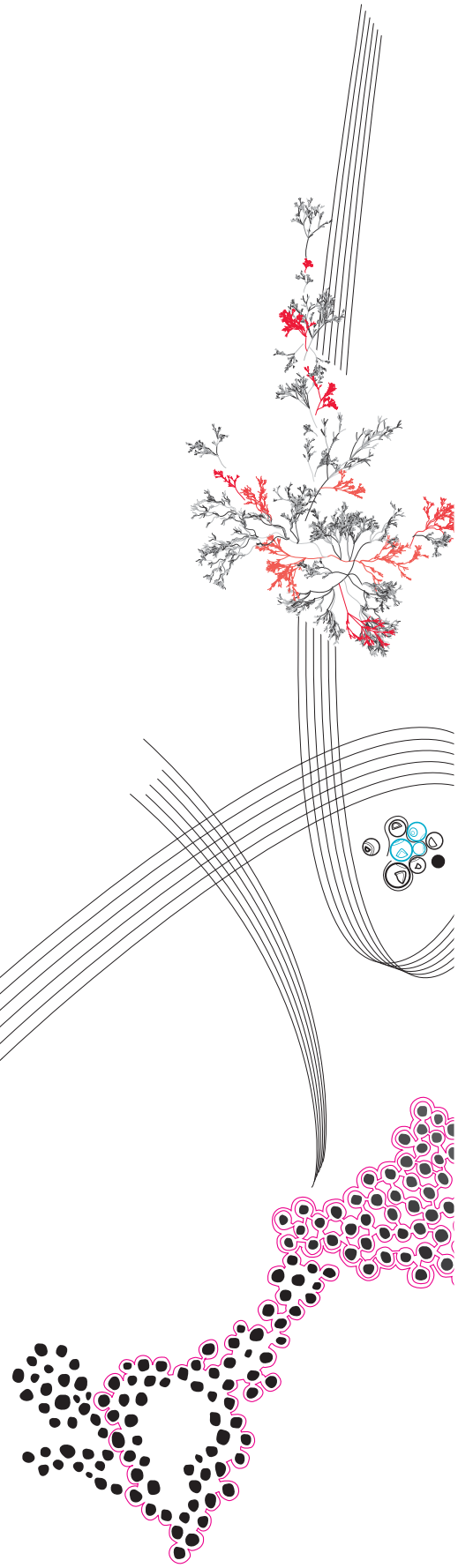


MSc Computer Science
Final Project

Using LLMs to aid developers with code comprehension in codebases

Koen Reefman

Supervisor:
Fernando Castor de Lima Filho, PhD
Kevin van der Vlist EngD MSc
ir. Robbert van Dalen

Committee:
dr.ir. Vadim Zaytsev
dr. Shenghui Wang

August 23, 2024

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Abstract

Developers spend significant amounts of time on comprehending code, with research suggesting developers spend up to 70% of their time on code comprehension. This is worsened further if there is a lack of proper documentation. With the help of Large Language Models, this could be sped up greatly. However, current research on this topic is limited in scope, mostly focusing on an educational context. In this work, the effectiveness of LLM tools is evaluated in a real-world setting, with the usage of locally running models. Results show that such tools can help developers quickly narrow down where to look in the code when solving tasks. Additionally, results show that developers can get a good low-level understanding of the code without the aid of an expert on the codebase. The nature of this research, with its usage of locally running models, allows for several directions for improvements in future work.

Keywords: Large Language Models, Code Comprehension, Open-Source Models, Llama3

Acknowledgements

I would like to thank Kevin van der Vlist and Robbert van Dalen for their daily supervision during this thesis, as well as their significant time investment when rating LLM responses. Secondly, I would like to thank my main supervisor, Fernando Castor de Lima Filho, for his supervision, as well as his time spent rating LLM responses.

Lastly, I would like to thank all participants of the user study, who spent about 90 minutes each interacting with the LLM tool.

Contents

Chapter 1

Introduction

A problem most developers will face in their career is onboarding on unfamiliar codebases. Newly onboarded developers are expected to get to know the codebase as soon as possible, so they can start contributing to it. However, comprehending codebases is difficult, especially if there is also a lack of proper documentation for it, and the previous developers of the code are no longer available to answer any questions about the code. In the context of this research, the definition of "comprehension" is interpreted as a person understanding the purpose of the code to such a degree they can implement new features or fix bugs in that code. Research has shown that developers spend significant amounts of their time on comprehending code [29], with one study showing developers spend up to 70% of their time on comprehending code [42]. The lack of proper documentation also has a negative impact on program comprehension time [73].

One possible solution to alleviating this problem could be the usage of Large Language Models (LLMs), which have seen a rapid rise in popularity over the last few years. Tools like ChatGPT [1] or GitHub Copilot [2] are already widely used, and researched. However, most research is focused on code generation rather than code comprehension. The focus of this research is on code comprehension; Instead of relying on documentation or asking questions to an expert on the codebase, developers will ask questions to an LLM-based tool in order to understand the code.

This research is performed at a large Dutch financial institution, at which there are many internal APIs, structured in a microservice architecture. Two of these internal APIs are used for the user study at the end, which amount to 80k lines of code overall.

1.1 Research Questions

To investigate the usefulness of using an LLM-based approach instead of more traditional approaches such as asking experts, the following research questions will be answered:

- RQ1: To what extent can an LLM-based tool help a developer understand a large codebase?
- RQ2: How do professional developers interact with an LLM-based tool in order to understand a piece of code?

1.2 Structure

After this section, the background of several important topics of this research will be discussed. This includes background information about machine learning, LLMs, and code comprehension. Chapter 3 discusses related work that is directly related to this research. The methodology of this research is discussed in chapter 4, followed by the results in chapter 5, together with the discussion in the same chapter. Lastly, conclusions will be discussed in chapter 6.

[1] https://chat.openai.com/
[2] https://github.com/features/copilot/

Chapter 2

Background

Recently, the popularity of LLMs have seen a rapid rise of popularity with the introduction of tools like ChatGPT[1] and GitHub Copilot[2]. However, the techniques behind these tools have existed for many years and have gradually evolved into what they are today. This chapter will take a look at the most important advances in the field of Machine Learning (ML) and will gradually work towards the State of the Art (SOTA). Secondly, there will be a look at the field of code comprehension; How this field has evolved over the years and what it looks like today.

2.1 Machine Learning

As mentioned before, the field of machine learning has existed for a long time before it became as popular as it is today. The first major influential experiment was the Georgetown IBM Experiment in 1954 [23]. This experiment was used to showcase the possibilities of automated machine translation by translating over sixty Russian sentences to English. This experiment, despite all of its flaws, marked the beginning of machine translation being seen as a research field worthy of financial investment.

After this, many different advances were made in the machine translation field which will not all be discussed in detail. The main focus of this chapter is to give background information on how the current state of structure of LLMs has evolved. The first major advancement after the Georgetown IBM Experiment was the introduction of the multilayer perceptron (MLP) in 1958 [54]. This network was, however, not yet a deep learning network. The first deep learning network was invented in 1965 by Alexey Ivakhnenko [24], sometimes also called the "father of deep learning"; This network was the first feedforward network, a type of network which inspired many of the networks that would follow.

Five years later, in 1970, the modern backpropagation algorithm was introduced by Seppo Linnainmaa [35]. This algorithm is used to compute the optimal parameters of a neural network. However, this algorithm wouldn't be used to train actual neural networks until 12 years later, in 1982 [70]. These years between 1974 and 1980 are also known as the first AI winter, which was mainly caused by the lack of computing power at that time. Even though the backpropagation algorithm was widely used in training NNs, there was a major problem with it called the vanishing (or exploding) gradient problem. This problem causes the hidden weights of a network to become so small that it can completely stop the network from training (or become so large that training becomes unstable). This problem was eventually solved in 1997, with the introduction of Long Short Term Memory (LSTM) networks [21].

Jumping ahead in time to 2014, when a method called 'attention' was introduced [11]. This mechanism simulates human attention by assigning different orders of importance to certain words in a sentence. These 'soft weights' for a sentence can be processed in parallel with the usage of transformers [66], which were introduced in 2017 by Vaswani et al. These transformers are a deep learning architecture that is widely used in the models used today, such as GPT (Generative Pre-trained *Transformer*).

[1] https://chat.openai.com/
[2] https://github.com/features/copilot

2.2 LLMs

Nowadays, LLMs are used everywhere. This section will highlight some of the most important ones, and discuss several aspects of these LLMs.

2.2.1 Open-Source vs Closed-Source

There are two different kinds of LLMs: open-source and closed-source models. Closed-source models usually outperform the open-source models, but often have some limitations. Most importantly, since closed-source models are often developed for commercial use, users cannot look into specific implementation details, such as the model architecture or training data. Another drawback to closed-source models is that they are often hosted by an external company, which might require you to get a subscription and would involve you sending possible sensitive data to a third party. On the other hand, there are open-source models. These models can often be run locally, removing the need to send sensitive data to third parties. However, these models do perform a bit worse than closed-source models most of the time. Additionally, you need a lot of computational power to run larger models locally.

2.2.2 Important Models

As mentioned in section 2.1, transformers are widely used in the architectures of LLMs today. In 2018, the BERT model was introduced by Devlin et al. [19], which is based on the transformer architecture. This model quickly became the baseline model in Natural Language Processing (NLP), due to its significant improvement in performance over earlier State of the Art (SOTA) models.

GPT-3 [17], perhaps the most well-known LLM, can be used for a wide variety of tasks such as code generation and code comprehension. The model, which was released in 2020, was trained on extremely large corpora of text, just like its predecessors, GPT [49] and GPT-2 [50]. Even though the model was newer at the time, this did not necessarily mean that it outperformed the SOTA for all tasks at that time. Fine-tuned SOTA models, as well as techniques like Retrieval Augmented Generation (RAG) [34] still outperformed GPT-3 on all tasks, with GPT-3 only performing similarly to these models on a few occasions [17]. Later research has shown that ChatGPT solves most problems quite well, but it loses to SOTA by anywhere from 4% to over 70% [30]. This makes ChatGPT a bit of a jack of all trades, but a master of none. However, because ChatGPT is a jack of all trades it can be used in a wide variety of settings. It can facilitate a developer's understanding of the code, due to its unique self-explanation capability. Additionally, the model can also adapt the expected outcome due to this.

More recent advancements in LLMs have introduced even better models. Two of the most notable begin Google's PaLM 2 [9], and GPT-4 [1]. These two models are relatively similar in terms of performance, with PaLM 2 being better in certain situations and GPT-4 performing better in others. One situation where PaLM 2 slightly outperforms GPT-4 is on reasoning tasks, for example the WinoGrande [55] benchmark. PaLM 2 achieved a score of 90.9 on this benchmark, with GPT-4 only scoring an 87.5. On the other hand, GPT-4 is superior when it comes to code generation tasks. On the HumanEval [18] benchmark, GPT-4 scored 67.0 0-shot, while PaLM 2 gets a score of 37.6 pass@1 on this same benchmark. So considering this, these two models are currently the best closed-source LLMs

available.

There have also been significant efforts to create open-source models. Some of the most popular models are the Llama family created by Meta. These models, however, don't perform as well as their closed-source counterparts, with Llama 2 (70b parameters) performing close to older closed-source models like GPT-3.5 and PaLM [65]. Llama 2 scores a 29.9 on the HumanEval benchmark, and a score of 80.2 on the WinoGrande benchmark. Meta's most recent model, Llama 3, reportedly outperforms its predecessor 63.7% of the time [6]. It's also significantly better at code-related tasks than its predecessor, scoring 81.7 on the HumanEval benchmark. Other examples of open-source models are some of the models by Mistral AI, such as Mistral and Mixtral. The Mistral model outperforms some of the smaller Llama models (7b & 13b), while having a third of their size thus being significantly more efficient [27]. In 2024, Mistral also released their Mixtral8x7b model, which outperforms the 70b Llama 2 model.

2.2.3 Model Size

Another important aspect of LLMs is the model size. Previous sections have already used the terms "7b", "13b", and "8x7b". These terms indicate the amount of parameters in a certain model. Generally, the more parameters a model has, the more complex data it can process. However, having a large amount of parameters make models more computationally expensive to train and deploy. The most advanced LLMs today have massive amounts of parameters, with GPT-4 reportedly having approximately 1.8 trillion [57].

2.2.4 Hallucinations

Despite the great performance current LLMs possess, they are still prone to hallucinating (i.e. stating incorrect or nonsensical facts as if they were the truth). Research has shown the larger models generally hallucinate less then smaller models [46], due to their larger training sets. These hallucinations can already be triggered by replacing just a few tokens in the input prompt [74], meaning a small typo in an input prompt could lead to the LLM generating incorrect information. Secondly, missing information can have a large impact on an LLM hallucinating. There are several methods of limiting the amount of hallucinations, such as using the aforementioned RAG, which uses external knowledge, to generate an snwer [25]. Another approach to mitigate hallucinations is to iteratively re-generate answers, based on external knowledge until a satisfactory answer is generated [26].

2.2.5 Types of Models

Three different types of llm models that are important to explain are the decoder-only model, the encoder-only model, and the encoder-decoder model. Firstly, the decoder-only model is currently the most popular, as it's widely used in generative LLMs. The decoder-only model takes a prompt as an input and outputs the next word in the sentence, which is a prediction. Encoder-only models, on the other hand output an embedding of the input. These types of models are often used in classification tasks. Lastly, there are the encoder-decoder models, which are a bit of a combination of both the previously mentioned models. These models take an input text, which they encode, and output another text. These models are often used for translation tasks.

2.2.6 Vector Similarity Search

The previous section mentioned the different types of models, this section will focus of the one particular usage of encoder-only models: vector embedding. Vector embedding is the process of embedding a sentence (set of words) into a vector representation. One example of such representations would be *Vector("King") - Vector("Man") + Vector("Woman")* resulting in a representation close to *Vector("Queen")* [41]. These models will be used to find similar snippets of code as the one put into it at the start. Vector embedding of words has a long history, with the usage of neural networks for this task stretching back to 1991 [39]. In 2000, Bengio et al. [12] produced a very popular model architecture for this, which would be followed by many others afterwards. More recently, Mikolov et al. released word2vec, a technique for "learning high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary" [40]. This is the first technique that can train on such large datasets. One more recent (and open-source) model for this is nomic-embed-text, which outperforms OpenAI Ada-002 and text-embedding-3-small on both short and long context tasks [7].

2.3 Code Comprehension

One of the largest parts of a developer's job is code comprehension. In order to fix bugs, implement new features, etc. the developer first needs to understand how the code works. Studies have indicated that developers spend anywhere from 35 all the way up to 70 percent of their time on comprehending code [29, 42, 73]. Several factors can also influence the time spent on code comprehension, according to Xia et al [73]. Xia et al. also found that senior developers spend significantly less time on code comprehension than junior developers, with senior developers spending an average of 44.43% of their time on code comprehension, while junior developers spend an average of 66.37% of their time for this. The phase of the project also has influence on this, with developers in the development phase spending less time on comprehension than developers in the maintenance phase.

The code comprehension field has already seen significant efforts in researching ways to make this more efficient. For over 30 years, there has even been a conference for this field in particular, called the International Conference on Program Comprehension (ICPC)[3]. At this conference, researchers present their latest advances in the field of code comprehension. Earlier research has shown that developers generally comprehend a codebase in one of two ways: top-down and bottom-up. The top-down method, which is often used by developers who are familiar with a program's domain [67], involves first forming an idea of what the entire system is supposed to do before focusing on understanding specific snippets of a program. Secondly, there is the bottom-up method, which is the opposite of the top-down approach. This approach has a developer first focus on understanding specific parts of the code before forming an understanding of the system as a whole. More recent research in the code comprehension field has focused on tracking developers' eye movements when comprehending code [53, 52]. This research has shown that developers pay more attention to certain parts of code over others. Method signatures, for example, are examined more than method invocations.

Another avenue of code comprehension that has been researched extensively is different ways of writing more comprehensible code and the effects certain code layouts have on the

[3] https://conf.researchr.org/series/icpc

readability. Research found that the readability of code is heavily influenced by factors such as variable naming [32, 62, 14], with bad variable names having a negative impact on the time it takes to comprehend a piece of code [58].

In order to measure the effectiveness of code comprehension methods and tools, there are several approaches for this. There is the approach where participants are asked to provide information about the code, i.e. explain the essence of the code. A second approach is asking participants for their personal opinions on the tool or method they interacted with. Then there is also the option to have participants interact directly with the code, using the tool, which is less common in research. Most research utilizes a combination of these approaches [47]. Additional metrics often used are correctness (e.g. check if the participant provided a correct answer to a question), and the time it takes to complete a task, which can be used to estimate the effort required to solve the task.

2.3.1 Code comprehension tools

Extensive research has been done on tools for code comprehension. There are many different tool available for different parts of code comprehension, like analysis, extraction, and presentation [60, 64].

The first part, analysis, mainly involves tools that analyse certain aspects of the code. For this, static analysis tools like SonarQube [4] can be used. These types of tools analyse your code and report on any issues the code might have - mainly focusing on code quality. Secondly, extraction tools are mainly used to understand the code on a deeper level. Tools in this category include parsers, which essentially break down lines of code into smaller pieces of information that the compiler understands. Lastly, presentation tools can help users get a better overview of the entire code they are working on. Tools in this category can include graph representations of the information flow of a codebase, which can help users understand better how data flows through their code.

Most modern Integrated Development Environments (IDEs) include features from each of these categories, making it a convenient experience for developers when comprehending code. Recent research has, however, highlighted a gap in current tools, particularly when investigating unfamiliar codebases [2]. This research included existing documentation inside users' IDEs to further help them in understanding the code, without the need to search for the right documentation themselves.

[4] https://www.sonarsource.com/products/sonarqube/developer-edition/

Chapter 3

Related Work

Previous chapter discussed discussed the past of several important aspects of the thesis, such as machine learning and code comprehension. This chapter will focus on recent advancements in these areas which have a direct impact on this research.

3.1 Code comprehension tools

Before addressing the recent advances in code comprehension using LLMs, there are a few notable tools for code comprehension that do not use LLMs. As mentioned in section 2.3, most modern IDEs have some kind of support for these kinds of tools. Perhaps the most notable IDE, Eclipse [1], provides APIs to facilitate this integration [71]. However, IDEs don't support every feature of comprehension tools. One tool that recently addressed these deficiencies is CodeCompass [48], which is a web-based static analysis tool for code comprehension of large codebases. This tool contains all kinds of information, from relations between classes or objects, to higher-level representations of the code, such as diagrams. Other research in this field includes the usage of different types of graphs to understand code, such as the call graph, which focus on a higher level understanding of the code. More low level graphs, such as code graphs [15], can also be used to understand code-level dependencies, declarations, etc.

3.2 LLMs for code comprehension

There have been a few recent advances in the field of code comprehension that are looking at using LLMs to help developers understand code faster. However, as far as we are aware, these studies solely focus on their usage in an educational context. Their usage so far has focused on helping students understand small code snippets and core concepts of these snippets [56, 33, 37]. Some of these studies have even shown that some students prefer machine-generated explanations over their human-generated counterparts [33].

One of the most important parts necessary for understanding code are the accompanying code comments. Several studies have shown that having code comments is beneficial for comprehending code [16, 63, 73]. However, in reality code comments might be incorrect/incomplete, or even be missing from the code altogether. There has been significant research efforts into automatically generating code comments in the past. Earlier efforts were mostly focused on filling in details in a manually created template [59]. More recent advances have shifted from the manual templates to explore neural network (NN) and LLM based methods [22, 75, 3, 72, 28, 61]. Earlier work by Hu et al [22] involved feeding a modified Abstract Syntax Tree (AST) into a sequence-to-sequence model, which then generated comments based on the AST structure. Later work by Ahmad et al. [3] used the same input, but fed this to a transformer model instead, improving the results even further. With the emergence of LLMs, however, research has shifted more towards directly using the source code as inputs. One example of this is Khan et al.'s work [28]. Since LLMs have been trained on large corpora, they are able to understand the given input and generate comments accordingly. The performance of LLMs can also be further increased using certain prompt engineering techniques, and even incorporating other data such as static analysis results in the prompt [5].

[1] https://www.eclipse.org

3.3 Prompt engineering

In order to get the best results possible from LLMs, several techniques have been invented to optimize the LLM's understanding of the input. These techniques are all used to give the LLM more context/ better structured context. The first technique that should be discussed is 0-shot, 1-shot and few-shot learning. This technique involves providing examples to the LLM that show how a response should be structured. 0-shot doesn't include any examples, one-shot includes one, and few-shot can include any amount more than one. Several studies have shown that few-shot learning delivers better results than 0-shot and 1-shot learning [17, 4], however this is not always possible due to there being no examples available, or the limited context window of some LLMs. Additionally, in order to get relevant examples, information retrieval techniques have to be used. [36, 44].

One more recent technique to improve results is chain-of-thought (CoT) prompting, which was introduced in 2022 by Wei et al. [69]. This technique involves asking the LLM to explain it's thought process step-by-step, instead of giving a straight answer. This greatly improved results, compared to normal prompting. Even more recently, research has shown that even better results could be produced by using self-consistency [68]. With this technique, the LLM generates answers to the same prompt multiple times (possibly using CoT prompting) and chooses the most common answer. This is a way to mitigate the amount of hallucinations produced by the model.

A different way to improve the LLM-generated answers is to include semantic facts about the code in the prompt. Earlier research on this in 2019 by Alon et al. focused on generating code summaries by using the AST of a particular piece of code [8]. Later research showed that using a data flow graph instead of an AST produced even better results [20]. These papers were however not using LLMs yet. Ahmed et al. investigated the usage of semantic facts in LLM prompts in 2023 [5]. The authors found that including these semantic facts did generally help the LLM and suggested it surpassed the SOTA.

3.4 In-IDE LLM usage

As mentioned before, the usage of LLM tools have seen an increase in popularity. Tools that can be used inside an IDE, such as GitHub Copilot[2] and Tabnine[3] are also used by an increasing amount of developers. However, these tools are generally used for code generation as opposed to comprehension. Recent research by Nam et al. [43] has focused on the usage of LLMs in an IDE for code comprehension, which found that it significantly increases a developer's ability to complete tasks compared to using traditional search-based information seeking.

[2] https://github.com/features/copilot
[3] https://tabnine.com

Chapter 4

Methodology

4.1 Soft-/Hardware

This section will discuss the various different hard- en software used for the experiments.

4.1.1 Hardware

As mentioned in chapter 2.2.1, there are open and closed source models. Since there are strict rules regarding data safety with the company that the research was performed at, closed-source models are out of the question. Additionally, using LLM as a service wasn't allowed due to computational costs. For this reason, only open-source, locally running models will be considered. It wasn't possible to get a cluster to run these models either, so they were all ran on a 2023 MacBook Pro with 32GB of RAM and an M2 chip.

4.1.2 Models used

LLMs will be used for two different tasks: vector embedding, and generating the actual response. In order to run the models, a program called Ollama[1] will be used. Ollama works by starting a server locally, which exposes several REST endpoints for the user to interact with. These endpoints are used to, interact with models, pull new models, push models, generate embeddings, etc. The minimal version for Ollama needs to be v0.1.31, since that version was the first to support embedding models. LLMs will be used for two different tasks: generating vector embeddings for an input code snippet, and generating the actual response. In order to generate vector embeddings, a comparison will be made between *nomic-embed-text*, *all-minilm:33m*, *gemma-7b*, and *codellama*. For generating the actual response, *mistral-7b* and llama3:8b will be compared. These models were chosen because they were the best open-source models available at the time.

4.1.3 Other Software

Along with Ollama, several other software will be used for different parts of the research. In order to store the vector embeddings generated with Ollama, a Chroma db[2] will be used. This database will be queried to find similar pieces of code to then one selected by the user. Since the tool developed during this research is a VSCode plugin, VSCode will also be included in this list. Lastly, Docker[3] will be used during the user study part of this research. Several Docker images will be created to more quickly set up different parts of the user study.

4.2 LLM Prompt

One of the most important things to consider when interacting with LLMs is the way the prompt is structured. In order to give the LLM as much information as possible to form a response, several different types of context have been included. These will shortly be explained in this section. Other forms of context, such as external documentation or the commit history are considered out of scope, in order to keep the focus of this thesis on undocumented code.

[1] https://www.ollama.com
[2] https://www.trychroma.com
[3] https://www.docker.com

4.2.1 Call Hierarchy

The first type of context that will be included in the prompt to the LLM is the call hierarchy. A call hierarchy shows all callers of a selected function, along with the callers of those callers. This will give the LLM context information about where a certain function is used, which it can use to form answers about the usage of a selected piece of code.

The call hierarchy is obtained using several built-in commands in VSCode. In order to get the call hierarchy of small parts of methods, the command *vscode.executeDocumentSymbolProvider* is used to get a *SymbolInformation* which gives information about the method which the selected code is in. This *SymbolInformation* is then converted into a *Position* in the file, which is then used to get the call hierarchy using *vscode.prepareCallHierarchy*. This call hierarchy is provided by the "Language Support for Java(TM)" plugin for VSCode, which provides Java language support via the Eclipse JDT language server.

To find the best way to represent the call hierarchy in the LLM prompt, two experiments will be performed.

Call Hierarchy Representation

Firstly, there are several ways to represent the call hierarchy in text form. For this experiment, four different representations will be examined. Below are examples of how the representations are represented in the prompt, based on the following call hierarchy, in which a is being called by b and c, and c is being called by d:

```
a
| b
| c
  | d
```

A nested list representation

```
[a <- [b, c <- d]]
```

Yaml-like indenting

```
a
    b
    c
        d
```

Grammar-like representation

```
a <- b
a <- c
c <- d
```

Grammar-like representation with lists

```
a <- [b, c]
c <- d
```

These four representations are all tested on the same piece of code with a sufficiently sized call hierarchy, found in figure 4.1 below. The LLM is asked "What methods call the selected method directly", for a total of 5 times for each representation. With this question, the response should only include the direct calls to the selected method and not include any deeper nested calls. For each response, the amount of tokens in the input prompt are also calculated, which will also be taken into account when drawing conclusions regarding the best way to represent the call hierarchy.

The results are analysed in two ways: by using vector embedding to compare the response to a ground truth and by manually reviewing each response to ensure the vector distance accurately reflects the quality of the LLM response. These results are presented in section 5.1.1

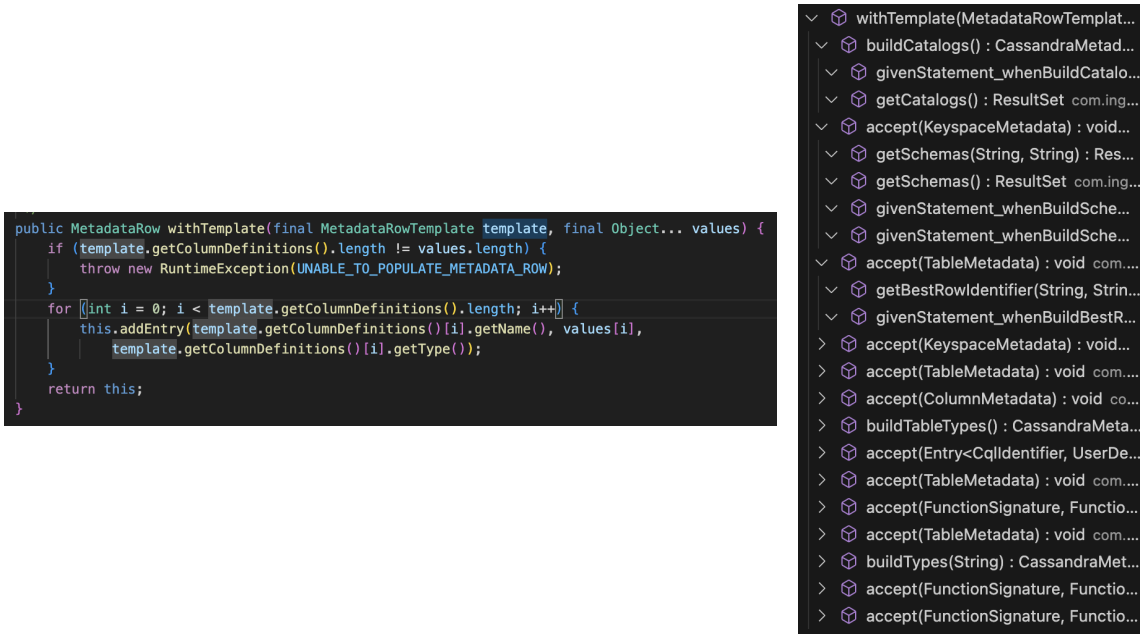


FIGURE 4.1: Code snippet and its corresponding call hierarchy

4.2.2 Similar Code

Secondly, the prompt will include similar pieces of code to the one selected by the user. The intention is to give the LLM some examples of similar pieces of code in order to form possible answers about any specific coding patterns used throughout the codebase. These similar code snippets are found by using vector similarity search, mentioned in chapter 2.2.6.

In order to find similar snippets of code, ElasticSearch[4] was considered, which is a sort of search engine you can put your own data into. ElasticSearch is built on top of Lucene[5] and uses the Okapi BM25 algorithm [51] to find similar results. However, since the Okapi BM25 algorithm ranks results based on the word frequency of documents, this results in a set of results that are semantically similar (i.e. have similar naming), and not necessarily all relevant syntactically similar (i.e. similar meaning/ structure of code). For this reason,

[4] https://www.elastic.co
[5] https://lucene.apache.org

and because Ollama had just started supporting vector embedding in version v0.1.31, it was decided to step away from ElasticSearch and use vector similarity search instead. For these experiments, as well as for general use, vector embeddings need to be generated for all source code files in a codebase. In order to store these generated vector embeddings of all the code in a codebase, a Chroma DB[6] is used.

The model chosen for vector embedding is nomic-embed-text. This model is, based on current research, the best suited for our purposes, as it's a small, fully open-source model that outperforms OpenAI's Ada-002 [45]. To get the best possible similar code snippets, experiments were done that looked at different chunking strategies. The first thing looked at was chunk overlap, which is the amount of tokens that overlap with a different chunk. Since it has been shown that an overlap of 25% is an effective split [13], this will also be used in this current research. Secondly, the chunk size itself was looked at. Generally, the smaller a chunk is, the more information it holds. Since the aforementioned article has also shown that a smaller chunk size generally holds more information and is thus 'better'. In order to keep the results manageable in size, in order to fit them into the prompt, a chunk size of 400 with an overlap of 100 was chosen. It should be noted that this is the **maximum** chunk size, chunks are also allowed to be smaller than this. The full source code files are all split into smaller parts using LangChain's RecursiveCharacterTextSplitter.from_language.

4.2.3 Repository Structure

The last piece of context put into the prompt is the repository structure. Including information about the repository structure allows the LLM to accurately refer to other parts of the code (for example when referring to similar pieces of code). The context includes the entire structure of the repository which the user is currently in. However, for very large repositories, this will lead to the prompt not being able to handle the amount of tokens necessary. The experiments done regarding the repository structure all look at the amount of information that is included in the structure. This includes in- or excluding the full filename from the prompt, in- or excluding the test directory of the repository, and only including parts of the repository structure that show up in the call hierarchy of a selected piece of code. The full table with combinations is found below (table 4.1).

Experiment ID	Filenames	Test Directory	Only CH
1	✓	✓	✗
2	✓	✗	✗
3	✗	✓	✗
4	✗	✗	✗
5	✓	✓	✓
6	✓	✗	✓
7	✗	✓	✓

Table 4.1: Repository structure experiments

These experiments aren't evaluated separately, but instead are evaluated in the experiments in sections 4.4 and 4.5. There, these different parameters are evaluated using vector similarity and human ratings, respectively.

[6] https://www.trychroma.com

4.2.4 Prompting techniques

Next to all the different types of context that are included in the prompt, several prompting techniques to improve responses have been looked at. The first technique used is giving the LLM an instruction, or giving the LLM a specific role it should adhere to. In this case, the LLM is being told "You are an expert developer who is tasked with onboarding a new developer on your codebase, only answer the questions asked by the developer. Keep your answers concise". The first part of this is used to indicate a specific role to the LLM, in this case "an expert developer who is tasked with onboarding a new developer". The second part is used to keep the LLM from generating too many things that aren't specifically asked by the user. The last part is used to keep the answers of the LLM short.

The second technique that was investigated was the difference between 0-shot, 1-shot and few-shot prompts. While it would have been better to use few-shot, or at least 1-shot, prompts; This was deemed to be infeasible very early in the project. In order to use multi-shot prompting, a curated dataset of question-answer pairs would be necessary to answer questions by the user. At the time of this research, as far as we are aware, such a dataset was not available. Secondly, including multi-shot examples in the prompt would also take up a portion of the limited amount of tokens available in the prompt with the current hardware limitations (4.1.1). For these reasons, it was decided to use 0-shot prompting instead and have the LLM learn in-context.

Another more recent prompting technique that wasn't used in this case is self-consistency. Having one prompt ran multiple times and picking the most common one was considered too time-consuming with the current hardware limitations in place. Since the user study involves running these models locally on the laptops of the engineers themselves, the answers would have to be produced reasonably fast. Lastly, one technique that was used is chain-of-thought (CoT) prompting. For this, the sentence "Let's think step by step" is added to the bottom of the prompt, which has been shown to improve results [31].

4.3 Tool Design

This section will describe the tool used for the user study, as well as explain how the different types of context discussed in the previous section are obtained and put in the prompt. The full source code of the tool, as well as a small demo, are available at `https://github.com/Koen-Reefman/llm-code-comprehension`.

4.3.1 User Interface

Figure 4.2 shows what the VSCode plugin looks like, each element in the figure is explained in detail separately. The simplistic design is loosely based on popular LLM tools such as ChatGPT in order to leverage the users' familiarity with similar tools. The choice was made to have the tool as an IDE extension to limit the amount of navigating the user has to do to ask questions about the code to the LLM.

1) Selected Code

In order to ask any questions about code, the user first has to select the piece of code which they want to ask a question for. This piece of code is put into the context, and is used to generate other parts of the context as well (call hierarchy and similar snippets).

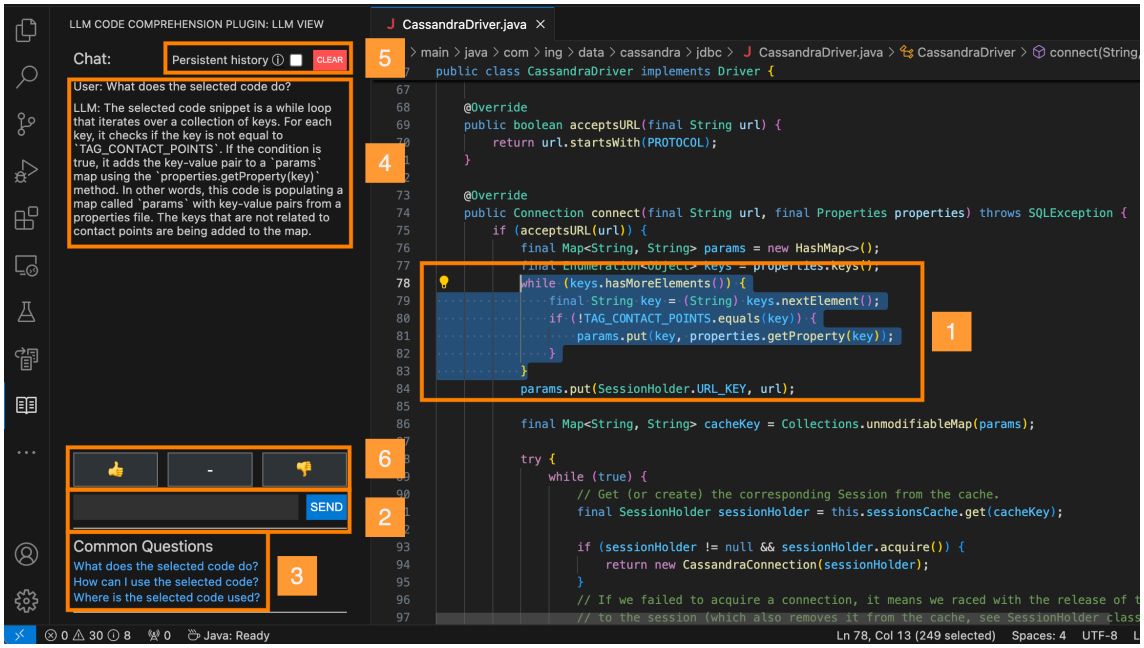


FIGURE 4.2: Overview of the LLM tool. (1) The selected code which is used for the context; (2) A user input box to ask questions; (3) Pre-defined questions; (4) The conversation history; (5) A persistent history toggle, and a delete history button; (6) Feedback buttons user study participants could use to indicate behaviour.

2) User Input Box

With a piece of code selected, the user can construct their own question about the code for the LLM, or use one of the pre-defined questions to get started.

3) Pre-defined Questions

At the bottom of the plugin, there are three pre-defined questions. Based on Nam et al.'s [43] findings, these questions were included as a starting point for users who are less experienced with using LLM tools and the way of constructing questions.

4) Conversation History

The full conversation history between the user and the LLM is shown. This allows users to refer back to older answers the LLM generated.

5) History Buttons

There is two different buttons that relate to history. The first is a red button to clear the conversation history with the LLM, so it won't refer back to older answers. The second is to keep a persistent history, even when selecting a different piece of code (thus refreshing the context). By default, new context is added to the prompt whenever a different piece of code is selected.

6) Feedback buttons

Lastly, there are feedback buttons. These buttons show up after the user asked a question and got a response from the LLM and can be used to indicate if the LLM's response is

good. Any feedback gotten from these buttons is saved during the user study part of this research.

4.3.2 Response Generation

In order to generate the LLM response using locally running models, Ollama is used to serve these models locally. Ollama exposes API endpoints locally, with which a simple HTTP request can be used to generate a response. In the code, LangChain's ConversationChain [7] is used, with a ConversationSummaryBufferMemory [8] to store the conversation history. This buffer memory first of all holds up to a certain amount of tokens in memory, and will summarize the history if the amount of tokens is exceeded. Since Ollama is just a locally running API, the models can easily be swapped out for other locally running models, or even non-locally running models such as OpenAI's models.

4.3.3 Prompt

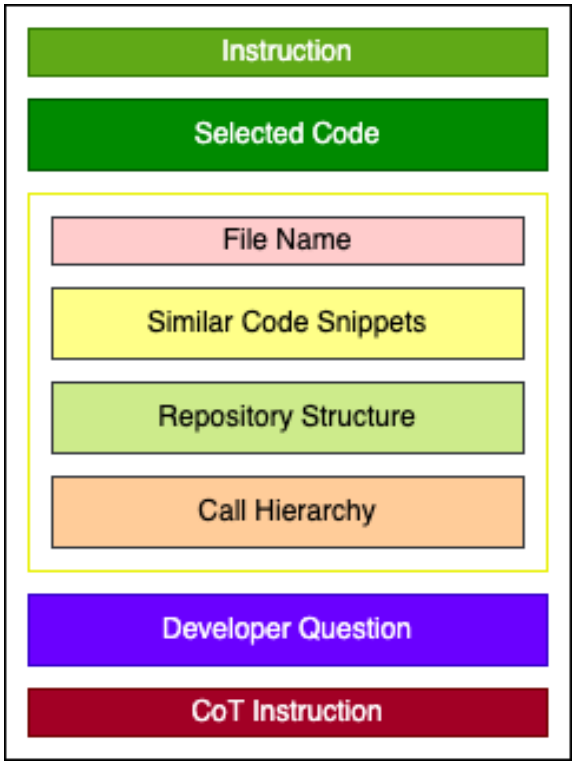


FIGURE 4.3: Schematic overview of the structure of the prompt

Instruction

The full prompt is first started with an instruction for the LLM. In this case, the instruction is: "You are an expert developer who is tasked with onboarding a new developer on your codebase, only answer the questions asked by the developer. Keep your answers concise". In order to limit the LLM as much as possible from generating answers irrelevant to the

[7] https://v02.api.js.langchain.com/classes/langchain_chains.ConversationChain.html
[8] https://v02.api.js.langchain.com/classes/langchain_memory.ConversationSummaryBufferMemory.html

question, it is asked to only answer questions asked by the developer, and keep the answers concise.

Selected Code & File Name

As mentioned in 4.3.1, the code the user selected is included in the full prompt, as well as the full location of the file in which the code is selected.

Similar Snippets

In order to get similar code snippets as the one selected by the user, vector similarity search is used. These similar snippets are searched by looking through a vector store, Chroma, and finding the most similar snippets based on a vector embedding of the input. The input is embedded using nomic-embed-text, which is once again running locally using Ollama. Beforehand, the entire codebase was already vectorized using nomic-embed-text, and the embeddings are stored in the Chroma database. There are two vector stores, one containing vectors of code snippets up to 400 tokens, and a larger one for snippets up to 1000. If the user selects a piece of code that's larger than 1000, first similar pieces of code to the vectorized input are searched, which are then split into smaller chunks in order to find better related similar pieces of code. The top K best results are returned, based on the maximum context size of the prompt.

Repository Structure

The repository structure of the codebase is obtained by calling the following VSCode function:

```
vscode.workspace.findFiles(**/*.java)
```

This returns a list of paths and files, which is then transformed into a JSON object. This JSON object is later modified based on experiment parameters found in section 4.5 (i.e. in-/excluding the file names, only including call hierarchy paths, in-/excluding test directory).

Call Hierarchy

The call hierarchy is obtained via the Java language server, by calling several VSCode commands. First of all, the start of the method needs to be found in order to get the call hierarchy, as calling the command in the middle of a function does not work. To do this, the start and end of a selected range is obtained, after which it is determined which methods intersect with the selected range by looking at the AST through the command:

```
vscode.commands.executeCommand(
    "vscode.executeDocumentSymbolProvider",
    editor.document.uri
);
```

After getting all methods that intersect with the selected range, the call hierarchies for all these methods is obtained using the following command:

```
vscode.commands.executeCommand(
    "vscode.prepareCallHierarchy",
    editor.document.uri,
```

```
    c.selectionRange.start
);
```

The results from this command are subsequently converted into the best representation, discussed in sections 4.2.1 and 5.1.1

Developer Question & CoT Instruction

The last part of the prompt contains the developers question in the format 'developer: question', as well as a small chain-of-thought instruction 'Let's think step by step', as mentioned in section 4.2.4.

4.4 Pre-experiment

Before performing the user study with participants, the best parameters/model would have to be chosen. For this, several experiments were executed in order to find the best performing parameters, which were compared to each other based on a set of constraints explained shortly. These pre-experiments were done before a subset of the total amount of experiment setups were evaluated by human reviewers, in the actual experiment.

4.4.1 Experiment setup

This pre-experiment is a large-scale automated assessment in order to limit the amount of responses that would have to be rated manually. This experiment consists of 44 different sets of parameters for 2 different models (Mistral and Llama3), with 3 questions being performed on 4 different code snippets. The parameters include several different configurations for four maximum context lengths: from 1k tokens all the way up to 8k tokens (which is the maximum context length for the mistral 7b v0.1 model which was used). For these different context lengths, each type of context gets an equal amount of the maximum context. For example, for the first experiment, the context size is 1024 tokens where all types of context get 200 tokens each. The remaining amount of tokens is used for other parts of the prompt (i.e. the first instruction, CoT instruction, user input, selected code, etc.). The code snippets used for this experiment are all found in cassandra-jdbc-wrapper, which is an open-source project with around 17k lines of Java code. Every experiment is ran 5 times to account for random variations with a temperature of 0.1 for each model. All in all, this leads to 44 * 2 * 3 * 4 * 5 = 5280 different LLM responses. The full set of experiment parameters, as well as all the questions and code snippets can be found in appendix B. A graphical representation of the total amount of experiment setups can be found in figure 4.4 below.

4.4.2 Rating / Exclusion parameters

In order to narrow down the best performing parameters for the different models, all responses of LLMs were compared to a set of ground truths. These ground truths are created based on the extensive comments found in the source code. For these ground truths, it is assumed that the comments in the code are correct, because it is an open source project, and open source projects are consistently well-documented [10]. This comparison was done by calculating the cosine vector distance between the LLM response and the ground truth. Below, there is a table with all the ground truths used, as well as a little explanation how the ground truths were created. The code snippets can be found in figures 4.5 to 4.7, below

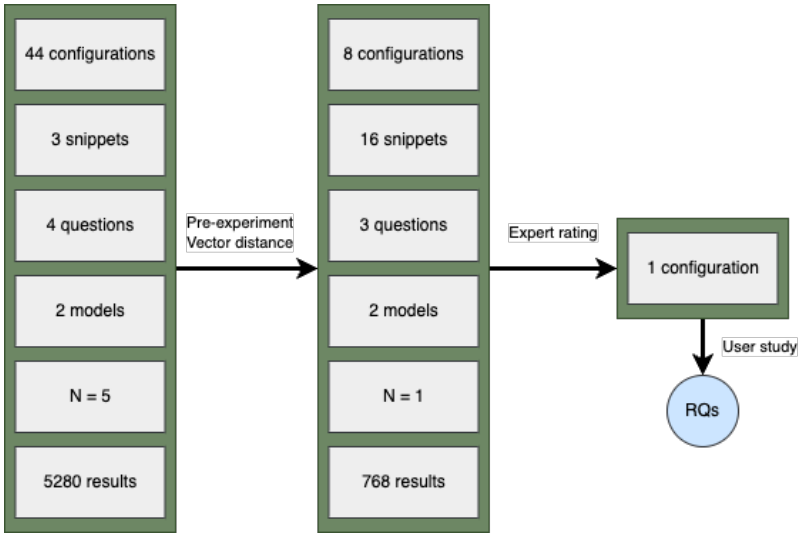


FIGURE 4.4: Amount of experiment configurations

the ground truths.

What does the selected code do?
The ground truths for this question are created based on the code comments that accompany the code snippets, as well as an interpretation of what the code does in one case (the third snippet, a unit test).

Ground truth snippet 1: *The selected code is used to populate a metadata row defined by a row template with the specified values. The number of values must match the number of columns defined in the row template, otherwise a runtime exception will be thrown.*

Ground truth snippet 2: *The selected method attempts to acquire a reference to this object and returns true if the reference was acquired successfully, false otherwise.*

Ground truth snippet 3: *The selected method is used to test if a result is returned when an SQL query with a vector table and similarity search is run.*

Where is the selected code used?
For this question, the ground truths are created based on the call hierarchy for the code snippets. These call hierarchies are inspected manually and put in the ground truth.

Ground truth snippet 1: *[buildCatalogs() (CatalogMetadataResultSetBuilder.java), accept(KeyspaceMetadata) (SchemaMetadataResultSetBuilder.java), accept(TableMetadata) (TableMetadataResultSetBuilder.java), accept(KeyspaceMetadata) (TypeMetadataResultSetBuilder.java), accept(TableMetadata) (TableMetadataResultSetBuilder.java), accept(ColumnMetadata) (ColumnMetadataResultSetBuilder.java), buildTableTypes() (TableMetadataResultSetBuilder.java), accept(Entry<CqlIdentifier, UserDefinedType>) (TypeMetadataResultSetBuilder.java), accept(TableMetadata) (TableMetadataResultSetBuilder.java), accept(FunctionSignature, FunctionMetadata) (FunctionMetadataResultSetBuilder.java)]*

Ground truth snippet 2: *[connect(String, Properties) (CassandraDriver.java)]*

Ground truth snippet 3: *The selected method isn't used anywhere, the test method is ran directly.*

How can the selected code be used?
The ground truths for this question are once again created based on the code comments, similarly to the first question.

Ground truth snippet 1: *The selected method can be used by creating an instance of a MetadataRow and calling the selected method with the required parameters.*

Ground truth snippet 2: *The selected method can be used by first creating an instance of a Sessionholder and then calling the selected method.*

Ground truth snippet 3: *The selected method is a test class and can directly be run within your IDE.*

What would an input to the function look like?
The ground truths for this question are created based on the code comments, as well as inspecting the method signature.

Ground truth snippet 1: *The selected method expects a MetadataRowTemplate and an amount of values used to populate the template.*

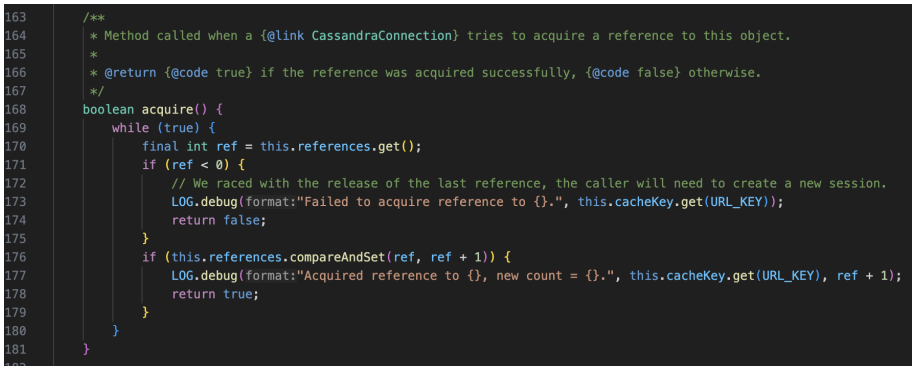
Ground truth snippet 2: *The selected method doesn't receive any input parameters.*

Ground truth snippet 3: *The selected method doesn't receive any input parameters.*

Additionally, an exclusion parameter was put in place for setups that took too long to generate an answer. Any response that took (on average) over 20 seconds to generate was automatically excluded, since it was deemed this was too long for engineers to wait for a response.

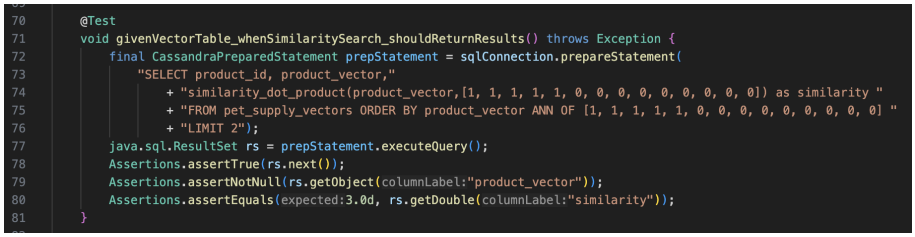
```
84      /**
85       * Populates a metadata row defined by a row template with the specified values.
86       * <p>
87       *     The number of values must match the number of columns defined in the row template, otherwise a runtime
88       *     exception will be thrown.
89       * </p>
90       *
91       * @param template The row template.
92       * @param values   The values used to populate the metadata row.
93       * @return The updated {@code MetadataRow} instance.
94       * @throws RuntimeException when the number of values does not match the number of columns defined in the row
95       * template.
96       */
97      public MetadataRow withTemplate(final MetadataRowTemplate template, final Object... values) {
98          if (template.getColumnDefinitions().length != values.length) {
99              throw new RuntimeException(UNABLE_TO_POPULATE_METADATA_ROW);
100         }
101         for (int i = 0; i < template.getColumnDefinitions().length; i++) {
102             this.addEntry(template.getColumnDefinitions()[i].getName(), values[i],
103                 template.getColumnDefinitions()[i].getType());
104         }
105         return this;
106     }
```

FIGURE 4.5: Pre-experiment code snippet 1



```java
/**
 * Method called when a {@link CassandraConnection} tries to acquire a reference to this object.
 *
 * @return {@code true} if the reference was acquired successfully, {@code false} otherwise.
 */
boolean acquire() {
    while (true) {
        final int ref = this.references.get();
        if (ref < 0) {
            // We raced with the release of the last reference, the caller will need to create a new session.
            LOG.debug(format:"Failed to acquire reference to {}.", this.cacheKey.get(URL_KEY));
            return false;
        }
        if (this.references.compareAndSet(ref, ref + 1)) {
            LOG.debug(format:"Acquired reference to {}, new count = {}.", this.cacheKey.get(URL_KEY), ref + 1);
            return true;
        }
    }
}
```

FIGURE 4.6: Pre-experiment code snippet 2



```java
@Test
void givenVectorTable_whenSimilaritySearch_shouldReturnResults() throws Exception {
    final CassandraPreparedStatement prepStatement = sqlConnection.prepareStatement(
        "SELECT product_id, product_vector,"
            + "similarity_dot_product(product_vector,[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]) as similarity "
            + "FROM pet_supply_vectors ORDER BY product_vector ANN OF [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0] "
            + "LIMIT 2");
    java.sql.ResultSet rs = prepStatement.executeQuery();
    Assertions.assertTrue(rs.next());
    Assertions.assertNotNull(rs.getObject(columnLabel:"product_vector"));
    Assertions.assertEquals(expected:3.0d, rs.getDouble(columnLabel:"similarity"));
}
```

FIGURE 4.7: Pre experiment code snippet 3

4.5 Experiment

4.5.1 Experiment setup

The pre-experiment narrowed down the amount of parameter setups that were considered good enough by a lot, to a total of 6. However, in order to find the actual best set of parameters, a second experiment was done in which experts rated the responses of the LLM. These human ratings also served a purpose in identifying how often the locally running smaller models hallucinated when generating their answers. In order to better rate the responses of each parameter setup, more different code snippets were added. The pre-experiment only included three, all of which were full methods. This experiment includes a total of 16 code snippets, including parts of methods as well. These parts of methods were included to better represent a real-world usage of the tool, in which engineers may have questions about small parts of a method instead of the full method. To account for the code snippets, the questions also had to be modified. Since the question "What would an input to the function look like?" does not make sense for parts of code that do not receive any input, this question was replaced by the question "What is being assigned to the variables in the selected code?". In the end, there were a total of 8 * 16 * 3 * 2 = 768 responses which needed to be rated.

4.5.2 Rating

In order to rate the responses, there were a total of three raters involved. Two of which are experts from the company the research is being performed at. In order to rate the responses more efficiently, a small Electron[9] program was created to include the information in a clear way. In this program, each rater would see the original question asked to the LLM, the piece of code which the question was asked about, as well as the response

[9] https://www.electronjs.org

generated by the LLM. The program also includes a link to the location of the selected code in GitHub, in case the reviewer would like some more information. Figure 4.8 shows an example of what a rater would see when rating responses.

Each response was rated on a scale from 1 to 5, with the ability for the rater to add

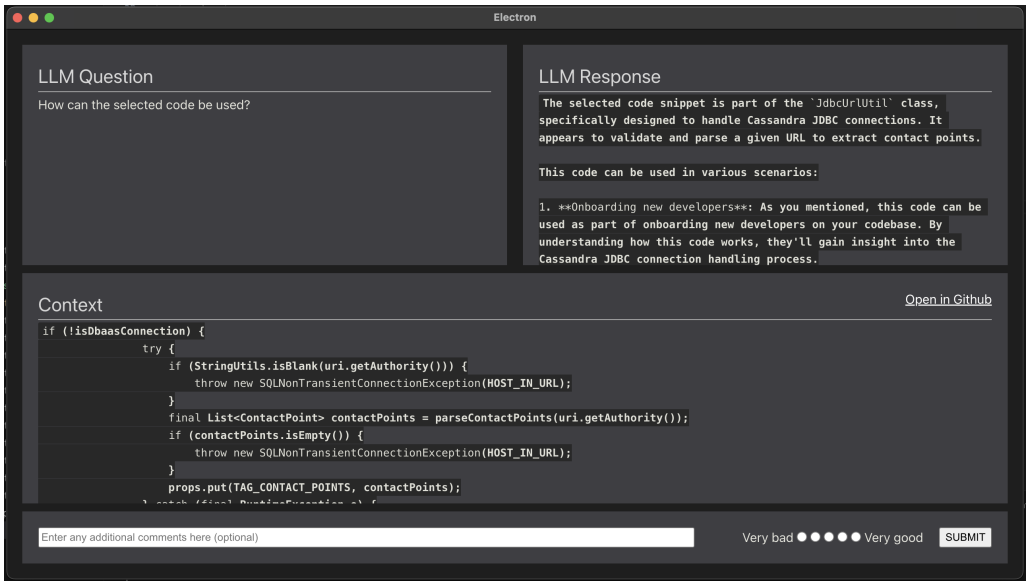


FIGURE 4.8: LLM response rater tool

additional comments to clarify their rating, if they wished to do so. Below, a table with an explanation for each rating is included.

Rating	Explanation
1	Incorrect information & hallucinations
2	Irrelevant information (e.g. types of variables that are explicit in the code)
3	Somewhat relevant information, which isn't necessarily helpful in understanding the code (e.g. types of variables that are inferred)
4	Useful information that helps understand the code
5	Useful information that helps understand the code both at a basic and more in-depth level

TABLE 4.2: Explanation of different response ratings

In order to reduce the load on individual raters, they were asked to rate 2/3 of the total amount of responses. The first person (an expert from the company) rated the first 2/3 of the responses. The second person (another expert from the company) rated the last 2/3 of the responses. The last person rated the first 1/2 and the last 1/3 of the responses. This lead to each response being rated by two people, and at least one expert. This way, each person had to rate 512 responses instead of all 768.

In the end, all ratings are aggregated and summarised. To determine the best set of parameters, several metrics are looked at:

Average rating
All ratings are summed for the different parameter setups (Llama3 and Mistral combined, as well as separately), which gives a good first indication which parameter setups perform the best.

Inter-rater reliability
For inter-rater reliability, Cohen's Kappa [38] will be analysed. For Cohen's Kappa, the ratings are "merged", meaning that ratings 1 and 2 are combined into a category "bad", and ratings 4 and 5 are combined into a category "good". In addition to the overall agreement, Cohen's Kappa will also be calculated for individual experiment setups, in order to investigate any correlation between different types of context and if raters agree on the responses for these setups.

Pearson correlation
Lastly, the Pearson correlation will be used to investigate if there is a correlation between the amount of tokens in a prompt and the perceived response quality of the LLM.

4.5.3 Major Disagreements

Due to there being some cases where raters strongly disagree (i.e. one reviewer voted a 1, while the other voted a 5), some cases were reviewed again by all three reviewers. In this small experiment, all reviewers were asked to give a comment as well, stating why they think the rating should receive the score they give. These results will not be used when determining the best set of parameters, but they will shortly be discussed.

4.5.4 Classification of Badly Rated Responses

In addition to having the raters re-rate the major disagreements, all cases in which one of the raters rated a response as 1 are evaluated and classified in different categories. This will give an impression what kinds of mistakes the LLM is most likely to make. These results are summarized and presented for future researchers to get an idea what types of mistakes should be addressed. Due to time constraints, the cases have only be classified by the two experts from the company this research was performed at.

4.6 User Study

During the experiments, the best model parameters are selected so the LLM responses are as good as possible. In order to answer the actual research questions, a user study will be performed with this selected parameter setup.

4.6.1 Structure

The user study consists of three parts: the pre-tasks survey, the tasks, and the post-tasks survey.

Pre-Tasks Survey

The pre-tasks survey is a short, 10 minutes, survey which is used to get some general background information from the user study participants and gauge their experience with certain tools and techniques. This includes questions about their experience with Java,

LLM tools, among others. Appendix H includes the full list of questions asked to the participants.

The Tasks

The bulk of the user study consists of a set of tasks on inner-source code from the company this research was performed at. In total, two codebases are used, for which two tasks each are set up for the participants to complete. As mentioned in the introduction, these codebases are both internal APIs part of a microservices architecture. They are both Java codebases, with 39k and 69k lines of code, respectively. The codebases were not shared with the participants beforehand, in order to prevent any preparation. The selected tasks are actual tasks which have been completed in the past and are typical tasks that engineers get when they onboard on these codebases. They have been carefully selected together with two senior engineers from the company to ensure that they are feasible to complete in the maximum allotted time of 60 minutes. Each individual task is shortly described below. These tasks have, however, been abstracted due to confidentiality restrictions from the company. There is also a small explanation for each task on how the participants should ideally solve the problem. The goal of these tasks is not to actually implement the necessary changes, but comprehend the code in such a way so it is clear what changes need to be made. Actually implementing these changes is out of scope, since they would take too long to implement in the given timeframe. Instead, the participant has to explain to the researcher what the problem is/ what changes need to be made in order to successfully complete the task. These answers are then compared to the solutions actually implemented in the relevant pull requests in these codebases. If there were any additional remarks about a task, the participants were asked to write these down as well. These answers given by the participants are also included in the results.

The participants are randomly split into two groups before completing the tasks. The first group completes task 1 and 2 without the help of the LLM tool, and solely uses the LLM tool to complete tasks 3 and 4. The second group does the opposite, only using the LLM tool for tasks 1 and 2.

Task 1
A Java NumberFormatException is being thrown when users put in a very large number in a String and try to parse it with `Integer.parseInt`. In order to fix this bug, the participant will have to change the type of some variables to a BigInteger so the large input value can fit in the variable.

Task 2
Making an API call returns a HTTP 204 code. At some point, the body of the response is used while creating a POJO, which will fail due to the fact that there is no response body for a HTTP 204. In order to fix this, the participant will have to do a check if the body is empty before creating the POJO.

Task 3
There is a certain API call in the code that returns a certain number of results, based on a parameter. A request was made to remove this limit on the amount of results. In order to solve the original issue, participants will have to remove this parameter from the HTTP request.

Task 4
There is a bug report of certain results from an API call being filtered out, while they should now be included in the result instead. The participants of the user study will have to add an extra clause in an if statement to stop these results from being filtered out.

In order to gauge the participant's understanding of the code related to the task they implemented, there is a quick questionnaire about the code the task was about. After each task, the participant is asked to explain the code they just worked on in detail, as well as answer questions about the difficulty of each task. These tasks can be found in appendix I. In addition to these questions, the time it took to produce a final answer is also recorded. Some answers to open questions will also be shown, but will not contain any mention of the source code due to the restrictions from the company.

Post-Tasks Survey

After all the tasks there is a post-tasks survey, which will take about 15 minutes to complete. This survey is used to ask about the participant's interactions with the LLM tool, the perceived usefulness of the tool, the tool's capabilities, etc. The full set of questions asked during the post-tasks survey can also be found in appendix I, below all the inter-task questions. The results also include the participants' explanations why they think this LLM tool is (or isn't) useful.

4.6.2 Participants

All participants that participate in the study work at the company at which the research is performed. Their levels of experience range from junior to senior engineers, with the slight majority being seniors. All participants have experience with programming in Java. Because the LLMs are being ran locally, some participants had longer wait times for LLM responses. To accommodate for this, a little extra time was allowed for those who felt the LLM's responses took too long to generate. There were a total of 7 participants. Unfortunately, getting more participants was rather difficult due to how busy the engineers were at the time of these user studies.

4.6.3 Data gathered

During the participants' tasks with the LLM tool, their interactions with it are monitored. This data will be analyzed in order to draw conclusions on how professional developers interact with LLM tools to understand a codebase. Since these interactions are done on inner-source code, they will not be included verbatim but will be paraphrased instead. For each interaction, the participant's question, the code they selected, and the LLM response are saved. In addition to this, participants also have the option to rate an LLM response. This allows participants to indicate any really good (or bad) answers, if they wish to do so. This interaction data will be analyzed in order to answer RQ2.

Chapter 5

Results & Discussion

This section contains the results of all different experiments done, as well as a discussion about the results accompanying them. In the first subsection, results from different representations of prompt parts will be addressed; The second subsection discusses the results from the automated pre-experiment to find the best set of parameters. After that, results from the experiment with three expert raters will be discussed, including major disagreements and classification of bad LLM responses. Lastly, all results regarding the user study will be discussed.

5.1 Prompt Parts

As mentioned in section 4.2, experiments were done with different types of context in order to have the LLM generate the best possible responses given the restrictions. This section will present the results from these experiments, as well as discuss the impacts they have on the LLM's responses, which will determine how the different types of context are represented in the prompts used during the (pre-)experiment.

5.1.1 Call Hierarchy Representation

As mentioned in section 4.2.1, the results are evaluated both automatically, based on a ground truth string, as well as manually. Table 5.1 shows the results of the automated evaluation. The vector distance in this table is the average over 5 total runs done with the same model (Mistral), with a temperature of 0.1. The full list of responses can be found in appendix A.

Representation	Average vector distance	Token amount
YAML	0,32085	627
Nested list	0,34811	930
Grammar	0,36323	1334
Grammar list	0,23822	509

TABLE 5.1: Automated assessment of different call hierarchy representations

The responses are also manually evaluated on their correctness, as a sanity check for the automated assessment. Below, there is a short summary of all the responses generated by the LLM for each call hierarchy representation. All responses are compared to the call hierarchy shown in figure 4.1. The full list of responses can be found in appendix A.

YAML

For the YAML representation, four out of five responses only included `buildFrom` and `MetadataResultSet`, both of which aren't in the call hierarchy to begin with. The fifth response does include items from the call hierarchy, but only includes 5 of the direct calls to the method.

Nested list

The nested list representation generally produces responses with more call hierarchy items than the YAML representation. Most items in the responses are also direct calls to the selected method. However, the LLM also includes items found deeper in the call hierarchy (`getSchemas`), as well as some items that aren't in the call hierarchy at all (`testIssue40`, `testIssue78`).

Grammar

In all five cases where the grammar representation was used, the LLM got the answer completely wrong and mentioned irrelevant items from the call hierarchy.

Grammar with list

Lastly, the grammar with list representation returns large amounts of call hierarchy items. In four of the five cases, the LLM only returned relevant items from the call hierarchy that directly call the selected method. In the fifth case (response 3), the response does only include items from the call hierarchy, but the LLM also explains items that are found deeper in the call hierarchy.

5.1.2 Discussion

Looking at table 5.1, the best way of representing the call hierarchy in the prompt is by using a grammar-like representation with lists to indicate multiple methods calling a single method. The table clearly shows this representation being closest to the ground truth, with the least amount of tokens used to represent it as well. Additionally, manually reviewing the responses also lead to the same conclusion of representing the call hierarchy with a grammar-list approach being the best. Overall, this representation allowed the LLM to generate the best responses, with some of the other representations (especially the YAML and grammar representations) producing sub-par results. These other representations also tended to mention non-direct calls to the selected method more often, which wasn't what the prompt asked for. The nested-list approach produced some decent responses to the prompt, but also had some situations where it mentioned non-direct calls to the method. For this reason, it was deemed to be too inconsistent to use, and the grammar-list representation was ultimately used in the prompts for subsequent experiments.

5.2 Pre-Experiment

Section 4.4 contains all snippets and their respective ground truths for all questions. Figures 5.1, 5.2, and 5.3 below show the results obtained during this experiment. There are four different smaller graphs in these graphs, representing different context sizes used. From left to right: 1k, 2k, 4k, 8k. The numbers on the x-axis correspond to the numbers found in tables C.1, C.2, and C.3 in appendix C. These tables contain the raw data of this experiment.

5.2.1 Spearman Correlation

Using figures 5.1, 5.2, and 5.3, the correlation between the tokens/prompt and tokens/distance is calculated. Using the formula $\rho = 1 - \frac{6\Sigma d_i^2}{n(n^2-1)}$, and the values from table C.4, the Spearman correlation between the amount of tokens in a prompt and average response time is $0.9497... \approx 0.95$. For Llama3 and Mistral separately, the Spearman correlations are $0.8965... \approx 0.90$ and $0.9574... \approx 0.96$, respectively. The Spearman correlation between the amount of tokens in a prompt and the average embedding distance from a ground truth is $0.4849... \approx 0.48$. The full tables for these calculations can be found in appendix C.

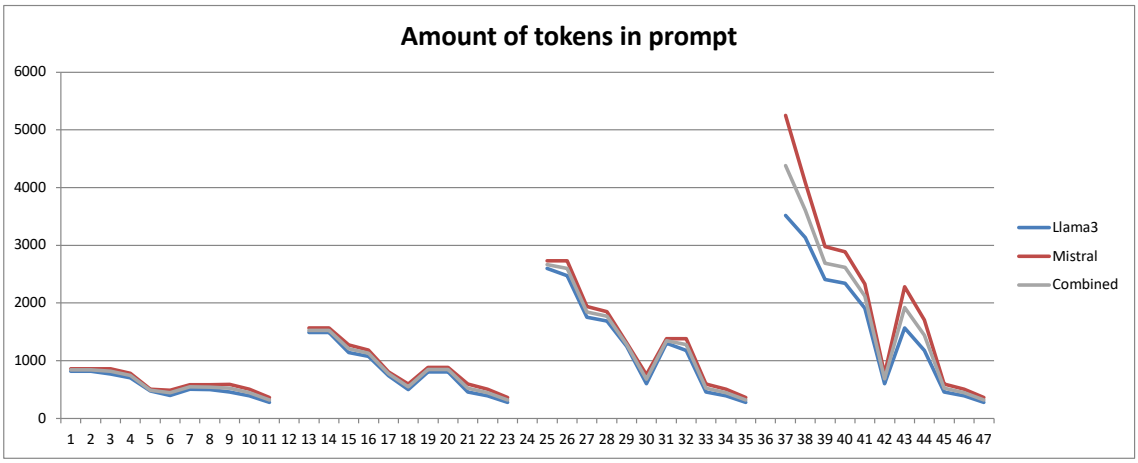


FIGURE 5.1: Pre-experiment amount of tokens in prompt

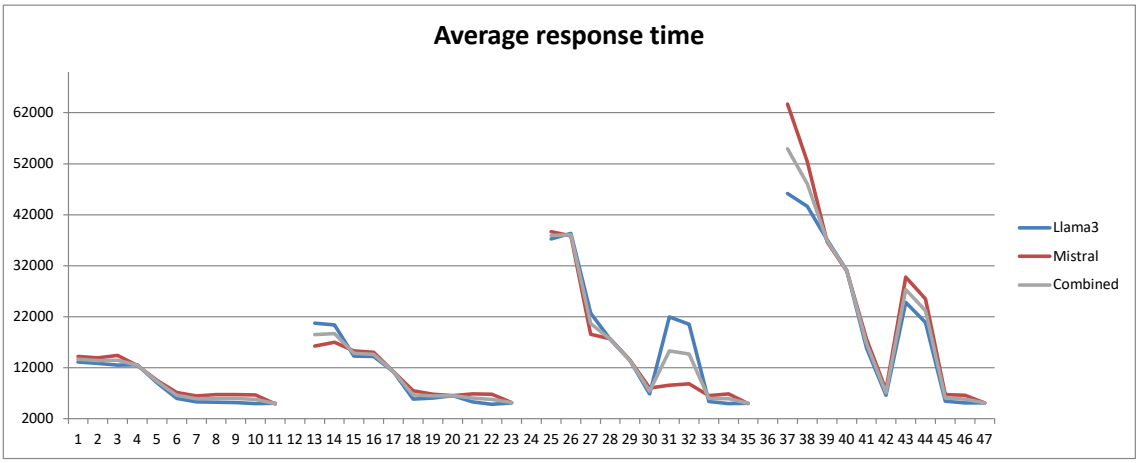


FIGURE 5.2: Pre-experiment average response time

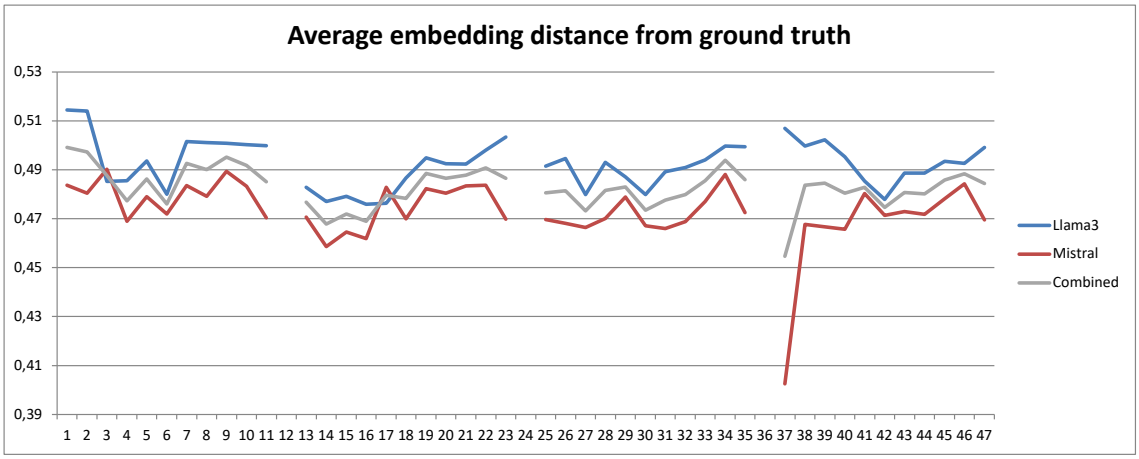


FIGURE 5.3: Pre-experiment average embedding distance from ground truth

5.2.2 Discussion

The aim of this pre-experiment was to find a subset of all experiment definitions, which would ultimately be used in the experiment, and be rated by human reviewers. Figures 5.1, 5.2, and 5.3 in the previous section show the amount of tokens in the prompt, the LLM response time, and the vector embedding distance from their ground truth, respectively.

These graphs show a clear correlation between the amount of tokens in the prompt and the LLM response time, but not necessarily in the quality of the results. This is also supported by the Spearman correlations found in the previous section, which shows an overall correlation of 0.96 between the amount of tokens and the response time. On the other hand, the overall Spearman coefficient for amount of tokens and average embedding distance is 0.48, which suggests there is a slight correlation between the amount of tokens in a prompt and the distance from the ground truth. One interesting thing to note in these graphs is the performance for the mistral model with full context (8k context window). This setup performed significantly better than any other, but it also took significantly longer to generate an answer.

Keeping in mind the exclusion parameter of excluding all experiment setups that took over 20 seconds to response, a subset of setups was chosen for the experiment. Table 5.2 below shows the chosen setups. A little plaintext explanation for each experiment is also provided in table 5.3. These setups are all from the smaller context sizes, since the 4k and 8k context sizes mostly took over 20 seconds to generate an answer, and the ones that were under 20 seconds contain little to no context (and can thus be fit into a smaller context size). Ultimately, the parameter setups chosen are the first six with a 2k context size (12-17 in the graph), and two of the best performing ones with a 1k context size (5 and 6 in the graph).

ID	Context Size	Similar Snippets	Call Hierarchy	Repository Structure	Use Filenames	Test Directory
1	1024	✓	✗	✗	-	-
2		✗	✓	✗	-	-
3	2048	✓	✓	✓	✓	✓
4		✓	✓	✓	✓	✗
5		✓	✓	✓	✗	✓
6		✓	✓	✓	✗	✗
7		✓	✗	✗	-	-
8		✗	✓	✗	-	-

Table 5.2: Experiment definitions

ID	Context	Plaintext explanation
1	1024	Only similar code snippets
2		Only call hierarchy
3	2048	All context, repository structure with file names and test directory
4		All context, repository structure with file names, without test directory
5		All context, repository structure without file names, with test directory
6		All context, repository structure without file names and test directory
7		Only similar code snippets
8		Only call hierarchy

Table 5.3: Plaintext explanations for experiments

5.3 Experiment

As discussed in section 4.5, this section contains results regarding the average ratings, inter-rater reliability, Spearman correlation, major disagreements, and classifications of bad LLM responses.

5.3.1 Average Ratings

Figure 5.4 shows the average ratings per experiment definition, rated by human raters. These results correspond to the experiment setups in table 5.2, which were the best performing setups in the pre-experiment. Plaintext explanations for these experiment setups can be found in table 5.3, as well as in appendix B, since these experiment definitions are a subset of the ones used in the pre-experiment.

In addition to the average ratings of each experiment setup, figure 5.5 below also shows how many of the ratings have received a rating of **at least** <x>. Figure 5.6 shows the distribution of ratings the raters gave to the LLM responses.

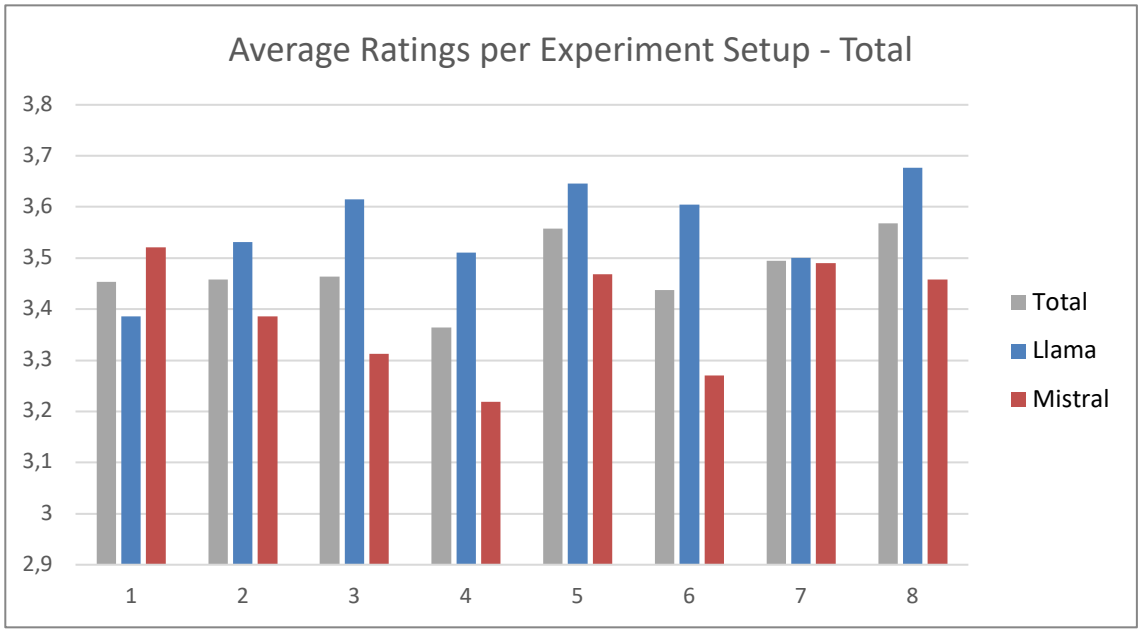


FIGURE 5.4: Total Results

5.3.2 Discussion

Looking at the overall ratings for each experiment setup in figure 5.4, the two best-performing setups are setup 5 and 8, with average ratings of 3.56 and 3.57 respectively. From the results, it can also clearly be seen that Llama produced better LLM responses than Mistral, which was expected since literature has also shown that Llama 3 outperforms Mistral. These results seem to imply that including the call hierarchy of a code snippet in the prompt is the most beneficial for the results, since experiment setup 5 (including all context) and 8 (only call hierarchy) both include the call hierarchy.

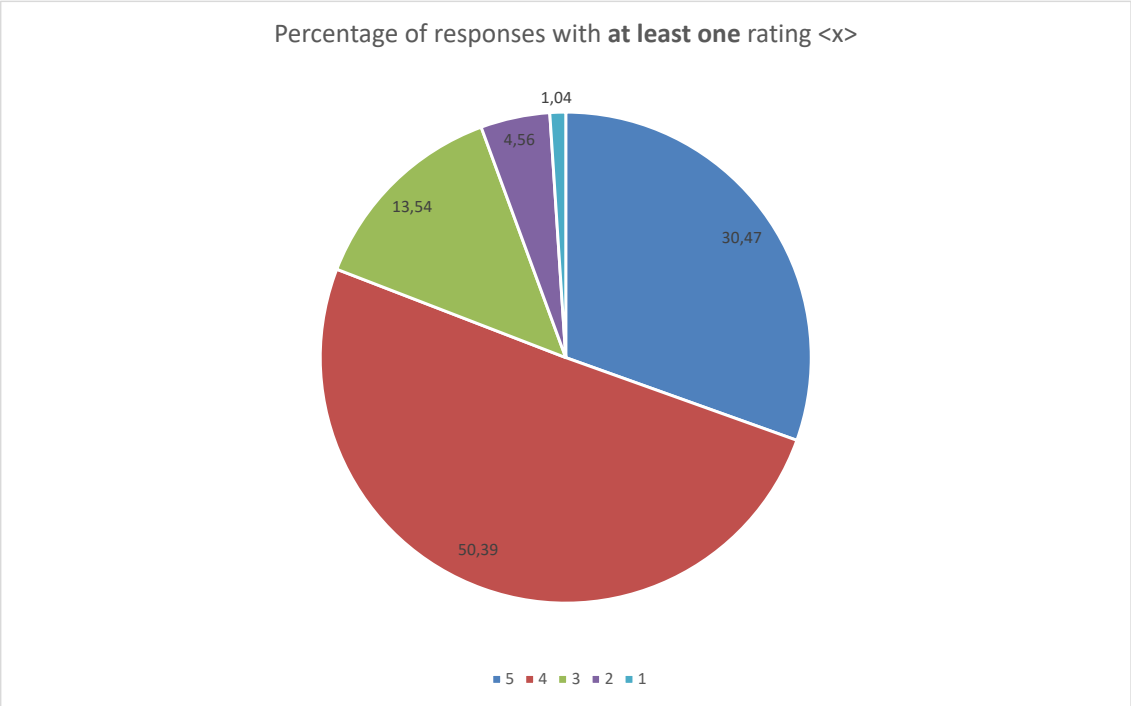


FIGURE 5.5: Percentage of responses with at least one rating <x>

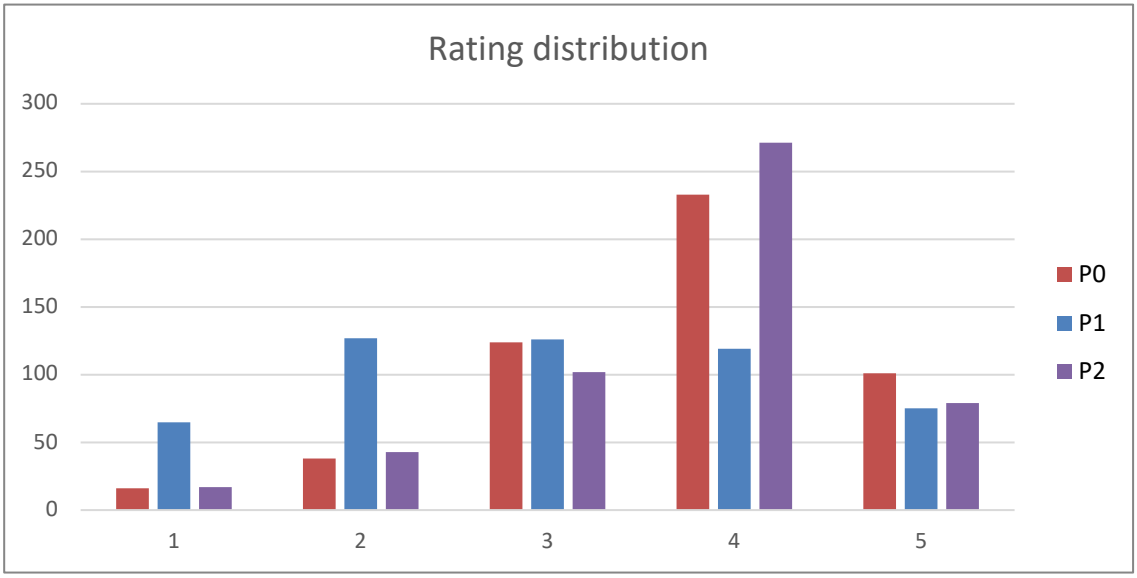


FIGURE 5.6: Rating Distribution

5.3.3 Inter-rater Reliability

In order to measure the extent to which raters agree when rating certain LLM responses, Cohen's Kappa is calculated [38]. Using the values from tables D.1, D.2, and D.3 in appendix D, Cohen's Kappa can be calculated between the individual raters. Cohen's Kappa, for all experiments, is ≈ 0.09 between P0 and P1, ≈ 0.08 between P0 and P2, and ≈ 0.05 between P1 and P2. The full calculations for these values can also be found in appendix D.

Agreement on individual experiment setups

The results in the previous section showed at the inter-rater agreement on all experiments as a whole, the results in this section show the agreement on different experiment setups. Table 5.4 below shows the different experiment setups, together with their average ratings and Cohen's Kappa between individual raters. All agreement tables, with which these values were calculated for can be found in appendix E.

Setup ID	Rating (average)	Kappa P0/P1	Kappa P0/P2	Kappa P1/P2
1	3,453125	0,011	0,241	0,038
2	3,458333	0,084	0,041	0,058
3	3,463542	-0,048	-0,176	-0,125
4	3,364583	0,227	0,118	0,165
5	3,557292	0,138	0,196	0,049
6	3,4375	0,162	-0,148	0,166
7	3,494792	0,034	-0,056	0,178
8	3,567708	-0,022	0,121	-0,075

TABLE 5.4: Cohen's Kappa for individual experiment setups

5.3.4 Discussion

On first glance, looking at figure 5.6, the ratings from P0 and P2 seem to be quite similar. This distribution indicates that P0 and P2 generally thought that the LLM responses were helpful in understanding the code it answered a question about. P1 appears to be more hesitant when rating the LLM's responses. However, when looking at the inter-rater reliability scores in the previous section, it's clear that the raters don't necessarily agree much. Interestingly, the highest Cohen's Kappa score is between P0 and P1, with a score of 0.09, which still indicates a low level of agreement. These low scores can be explained by the subjective nature of judging all these results, each rater has different opinions about the way LLMs formulate answers. Especially with the limited models used in this study, the model might generate an answer which is generally correct but contains a minor error. One reviewer might think this answer is correct and could be helpful in understanding the code, while another might think this small error could send users the wrong way when trying to understand the same code. One example of this is shown below:

Question: `How can the selected code be used?`

LLM Answer:

```
The selected code is part of a PooledCassandraConnection class,
    which seems to manage Cassandra connections and statements.
    The specific piece of code checks if an exception occurred
    during query execution. If it didn't occur (i.e., the
    exception is not a SQLRecoverableException), it closes the
    prepared statement and removes it from a used statements
    collection.

This code can be used in scenarios where you need to handle
    exceptions differently based on their type. For instance, if
    you're dealing with Cassandra queries that might fail due to
    temporary issues (like network connectivity problems), you
    might want to retry the query or log the error instead of
    closing the statement immediately.
```

Reviewer 1 (rating: 2):
"IT's sort of right, but it doesn't properly involve the context that well (i.e., that this is logic used throughout the code for the given purpose)"

Reviewer 2 (rating: 4):
"The explanation is not 100% precise in that it is not about an exception not occurring, but that specific type. The explanation in the second paragraph is pretty nice, though, as it goes beyond what's in the code and directly answers the question."

One threat to validity would be the limited amount of raters to rate the LLM responses. Since there were only 3 human raters, each rating 512 responses from an LLM, fatigue could have played a role when rating the LLM responses. This was slightly mitigated by having the raters rate small pieces of the total at the time (1/3 of the total amount, then have a few days off), but this can never fully be mitigated with this amount. Additionally, since each reviewer reviewed such a large amount of snippets, there was a risk of recognition bias: the more often a reviewer saw an explanation for the same piece of code they've seen countless times before, the more lenient they might be when rating the LLM's answer. This issue was slightly mitigated by increasing the amount of different code snippets compared to the automated pre-experiment, but could not fully be mitigated. There is also the issue of personal biases when rating responses. As shown in figure 5.6, raters P0 and P2 generally were more positive about the LLM, with P1 being a bit more negative. This could be due to different ways of interpreting the results - with raters P0 and P2 being more favourable to answers that generally send you in the right direction, and P1 being focused more on the overall correctness of the response.

Even though these threats regarding inter-rater reliability exist, there is still an overall sense of positivity towards the LLM's responses. As can be seen in figure 5.5, 80.86% of all cases were rated with a 4 by at least one rater. This indicates that at least one rater thought the answer was helpful in understanding the code snippet.

5.3.5 Spearman Correlation

The correlation between the size of an LLM prompt (in tokens) and the average ratings given by the raters is calculated using Spearman Correlation. Calculating this gives a value of 0.10. For the full calculation of this value, please refer to section D.2 in appendix D.

5.3.6 Discussion

A Spearman correlation value of 0.10 indicates there is no strong correlation between the LLM's prompt size and the average rating given by the raters for each exeriment setup. This suggests that adding more context doesn't necessarily improve the perceived results that the LLM produces. As mentioned in section 5.3.2, the type of context included has a larger impact. However, this also does not mean that the inverse is true, removing context certainly has a negative impact on the generated responses; As discussed in section 2.2.4, a lack of context leads to the LLM hallucinating more often.

5.3.7 Major disagreements

The full rating tables are shown in tables 5.5, 5.6, and 5.7. The total amount of major disagreements (meaning one rater rated 1, while the other rated 5) is 11. As mentioned in section 4.5.3, each rater was asked to re-rate the major disagreements. The original ratings for these major disagreements can be found in table G.1, while the updated ratings with explanations can be found in table 5.9. All LLM responses for these cases can be found in appendix F.

P0\P1	1	2	3	4	5
1	2	1	2	1	0
2	7	4	2	1	0
3	6	15	17	14	9
4	15	22	39	28	17
5	2	12	10	14	16

TABLE 5.5: Full rating table P0/P1

P0\P2	1	2	3	4	5
1	1	2	3	4	0
2	1	3	5	12	3
3	0	5	14	35	9
4	2	9	14	65	22
5	4	0	14	26	3

TABLE 5.6: Full rating table P0/P2

P1\P2	1	2	3	4	5
1	5	9	8	7	4
2	1	7	13	40	12
3	1	4	9	30	12
4	1	3	14	31	12
5	1	1	8	21	2

TABLE 5.7: Full rating table P1/P2

	P0		P1		P2	
ID	Rating	Explanation	Rating	Explanation	Rating	Explanation
1	-	-	5	-	1	NULL_KEYWORD is not a keyword, it is a constant.
2	5	this one includes the context very well, i like it	1	no not for testing purposes	-	-
3	5	-	-	-	1	No, no assignment is going on.
4	5	-	-	-	1	As far as I understand, this method is not about only one thread having access to the session. Otherwise, why have a reference counter?
5	-	-	1	onboarding process reference, too wordy - repetitions	5	-
6	5	-	1	return value is not a variable	-	-
7	5	-	-	-	1	Nothing is being assigned to columnIndex. Same for cqlDataType. DataTypeEnum.LIST is not even a variable, strictly speaking.
8	-	-	1	leaks **CORPORATE_KEY**. the ultimate caller <- wrong. weird details about Data, Time etc	5	Maybe the provided info is not incorrect, but it is so decontextualized that it ends up confusing more than enlightening, e.g., it mentions exceptions being thrown but nothing like this appears in the snippet, Same for data type comnversion. This does not seem to be what the method does.
9	-	-	1	leaks **CORPORATE_KEY**. verbose. not correct.	5	-
10	-	-	1	hard to follow, difficult structure, context leaking	5	-
11	5	-	-	-	1	Refers to three variables that do not receive assignments or based on which modifying operations are performed.

TABLE 5.8: Initial major disagreements

ID	**P0** (Re-)rating	Explanation	**P1** (Re-)rating	Explanation	**P2** (Re-)rating	Explanation	Average
1	5	Rating 2 is technically correct, but I personally don't mind as it is clear it should be a named constant. Rest is great so /care	3	not sure about this one. i think it is ok, but would rather have the LLM say constant rather than keyword	2	Not very informative. NULL_KEYWORD is not a keyword.	3,333333
2	5	I answered incorrectly. Just checked and format is indeed primarily used in testing.	3	primarily used for testing purposes - i'm don't think so Note that this time the LLM says 'predefined constant' which is correct.	4	It is not used mainly for testing purposes, but this is part of the context we've given the model. Could be addressed with some post-processing.	4
3	4	The text is completely wrong on the details, but the conclusion still holds 'effectively casting ...)	1	the assignment is happening....	1	Yes, there's no assignment and even if we try to extent the meaning of assignment, it still falls short.	2
4	3	True, it's not about exclusive access; https://t.ly/Bylv8	1	indeed, not about one thread having acces to the session - so it is wrong	1	Information is factually incorrect. It leverages reference counting.	1,666667
5	1	The onboarding reference makes no sense at all.	3	'your onboarding process'.	5	In spite of the context leaking, I think it directly answers the question and provides information that is not immediately obvious from the code.	3
6	5	This is an example of Robbert/Kevin (me) different approach: technically not an answer to the question (return value is not an assignment) but it is very useful.	2	still think return variable is odd	3	The feedback is being a bit misleading. Although it is potentially useful and informative, it does not answer the question. Furthermore, the description does not bring anything to the table that is not evident from the snippet.	3,333333
7	5	This is an example of Robbert/Kevin (me) different approach: technically not an answer to the question (return value is not an assignment) but it is very useful.	2	the question 'being assigned' is ambiguous, it could either be 'going to be assigned' or 'already assigned'. Also point 8 is not an assignment (it is a return value)	2	It provides a lot of information, but part of the information is misleading. Two variables it mentions are not even being assigned values.	3
8	1		2	the ultimate caller(s) are not only test classes. The details on Date, Time etc are not relevant	1	This comment does not even seem to refer to the code. Is it correctly matched to the snippet? In case it is, this should be a clear 1.	1,333333
9	4	I don't see what's not correct, but it is definitely verbose and leaks the corp key	1	context leaking \"similar code snippets', 'call hierarchy', '**CORPORATE_KEY**', the code is not about acquiring the same instance of SessionHolder	1	This seems to be incorrect. This snippet does not ensure mutual exclusion, as the explanation suggests (\"the first thread to successfully....\"). It is a ref counter, which means that multiple threads may acquire the resource.	2
10	4	Lengthy/verbose but seems to be correct. I don't really like the try-catch block hypothesis	2	the focus should be about assignments, not what the code does	4	Informative and covers all the variables, including 'data' (which does not get an assignment, but points to a buffer in the heap which is modified). The context leaking is bad, but minor. Could be removed with some post-processing.	3,333333
11	4	Lengthy/verbose but seems to be correct. I don't really like the try-catch block hypothesis	2	it should explain the variable assignments, not about the code in general	4	I've already commented on this one. "	3,333333

TABLE 5.9: Results from re-rating the major disagreements

5.3.8 Discussion

The overall positive sentiment uncovered in the previous section is also shown from the small amount of major disagreements there are. From a total of 768 cases, the reviewers only completely disagreed on a rating 11 times. After all reviewers re-rated these LLM responses, there was only one case left where the raters completely disagreed - ID 5. In this case, the LLm prompt leaked into the answer a bit, mentioning "your onboarding process". P0 thought this made no sense, thus giving it a 1. P1 also noticed this mention of the onboarding process, but decided to still give it a 3 since it could still be useful. Lastly, P2 thought the answer directly answered the question, despite leaking the prompt, and gave it a 5. In all other cases, raters' ratings were closer to each other. This change in ratings is most likely caused by the fact the raters didn't suffer from recognition bias or fatigue for the re-rating, as it was done a few weeks after the initial ratings. In 7 out of the 11 cases, the average rating for these responses was also 3 or above, indicating a more positive disposition to LLM responses when the raters weren't fatigued/recognizing the response.

5.3.9 Classification of badly rated responses

In total, there are 90 instances where at least one rater rated a response with a 1, out of a total of 768 cases. For the classification of the badly rated responses, the two raters who classified all responses agreed on 76 of the 90 cases. The other 24 cases, they didn't fully agree on how it could be classified. Figure 5.7 shows these 76 classifications. As can be seen from this figure, over 50% of the time, a bad LLM response is classified as "incorrect", with another 30% being classified as prompt leakage. Interestingly, the responses were only classified as "hallucination" 5 times, with the LLM mostly hallucination new variable names that don't exist in the prompt. The full set of classifications, along with the raters' justifications can be found in appendix G.

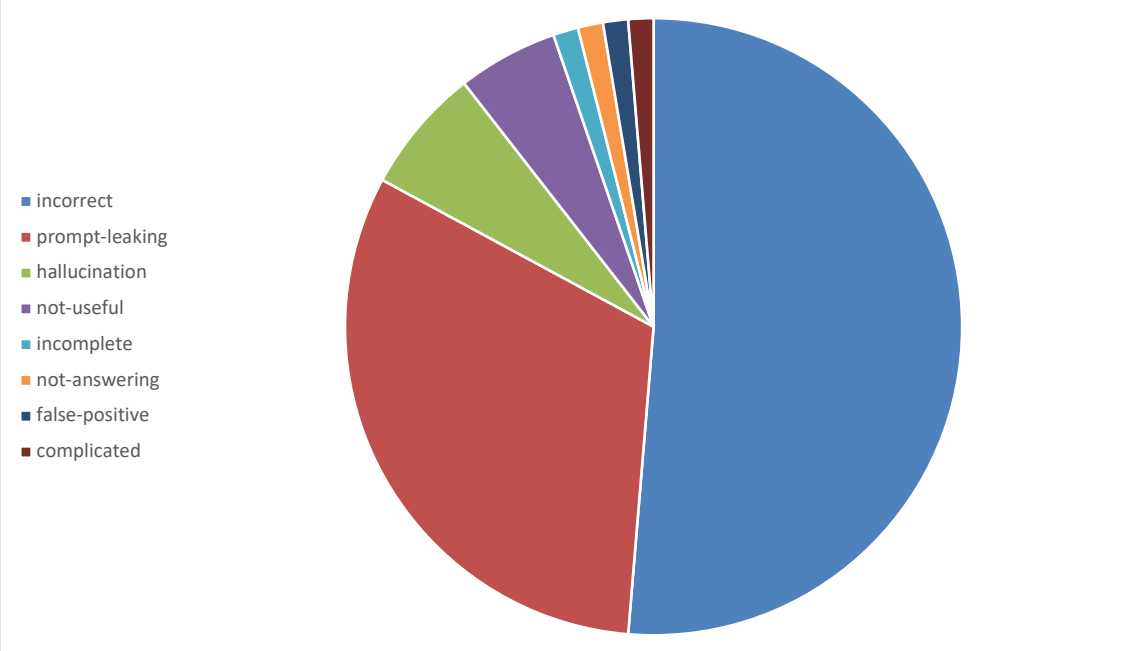


FIGURE 5.7: Classification of badly rated LLM responses

5.3.10 Discussion

As can be seen from figure 5.7, incorrect responses and prompt leaking are by far the most common mistakes the LLM tends to make. This result is likely caused by the fact there is so much context put into the prompt that is being fed to the LLM. Due to there being too much information, the LLM gets confused and generates false answers. Similarly, due to there being too much context in the prompt, the LLM loses it's focus (i.e. the instruction given to it at the start of the prompt - "only answer the questions asked by the developer") and talks about different parts of context mentioned in the prompt. On the contrary, due to the prompt containing so much information, the LLM doesn't tend to hallucinate new things often, which is good. All things considered, these results seem to show that the LLM used got a bit too much information for it to handle fully.
Even though the results suggest that the LLM only hallucinated five times, it should be noted that there is a very fine line between "incorrect" and "hallucination". In some cases, the raters also indicated that a response could be either of them. One such case where both raters rated the response as "incorrect", but mentioned a hallucinated variable is the following:

LLM answer:

```
In the given Java code snippet, the `close()` method of the `
    ManagedConnection` class is being discussed. This method iterates
    through all the statements associated with the connection and
    closes each one if it is not already closed. After closing all
    statements, it sets the `pooledCassandraConnection` and `
    physicalConnection` variables to null.

1. The variable `this.statements` is an array list of type `Statement`.
     It holds all the open statements associated with this connection.
2. The variable `statement` is of type `Statement`. It is used as a
    loop variable in the for-each loop to iterate through each
    statement in the `this.statements` array list.
3. The variable `isClosed` is of type boolean and is set to true once
    all statements have been closed.
4. The variable `this.pooledCassandraConnection` is an instance of a
    class that implements the Cassandra connection pool interface. It
    holds the underlying Cassandra connection pool.
5. The variable `this.physicalConnection` is an instance of a class
    that implements the Cassandra connection interface. It represents
    the physical connection to the Cassandra cluster.

These variables are assigned within the context of the `
    ManagedConnection` class.
```

Raters' classifications:

Rater	Explanation
P0	It's very wordy and imprecise. For example, 3. 'the var isClosed is bool and set to true once all statements have been closed' is bordering hallucination. Also some of them are not answers to the question
P1	'is being discussed' <- weird. isClosed is not a variable.

TABLE 5.10: Raters classifying a case as "incorrect", while mentioning hallucinations

5.4 User Study

As mentioned in section 4.6.2, a total of 7 participants participated in this user study, of which three of them are junior engineers and four are more senior, as shown in table 5.11. Unfortunately, getting more participants to complete a user study that could take up to 1.5 hours was difficult due to potential participants being too busy with their jobs. This could be one potential concern for the study. In order to mitigate this concern as much as possible, only participants who onboard in new codebases often were selected, so they could compare their usual experience with using the LLM tool.

5.4.1 Pre-Task Survey

Below, the results of the pre-task survey are shown. These results can be used to get an overview of the participants who took part in the user study. Table 5.11 contains data about the participants' professional programming experience, and the frequency of them onboarding on codebases they haven't seen before. Figure 5.8 shows data on what methods participants usually use when onboarding on a new codebase. In this image, one participant chose "other" with their answer being "Swagger documentation". Figures 5.9 and 5.10 show the participants' experience with using LLM tools for code-related tasks. Lastly, all participants in this study have experience with programming in Java, and most of them are VSCode users. One participant hadn't used VSCode before, so they were given a quick introduction.

ID	How many professional years of experience do you have as a developer?	How often do you onboard in a new codebase?
1	20	A lot
2	1	Once every three months or so.
3	9	once every 6 month
4	2	Once every 3 months
5	1	Once every 3 months
6	10	Once every month
7	6	A few times a year

TABLE 5.11: Pre-Task Survey results

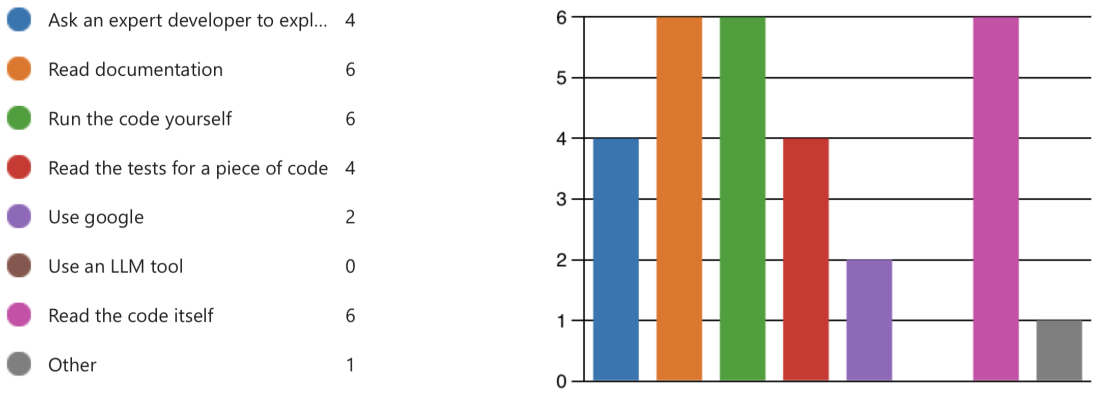


FIGURE 5.8: Participants' methods of understanding a new codebase

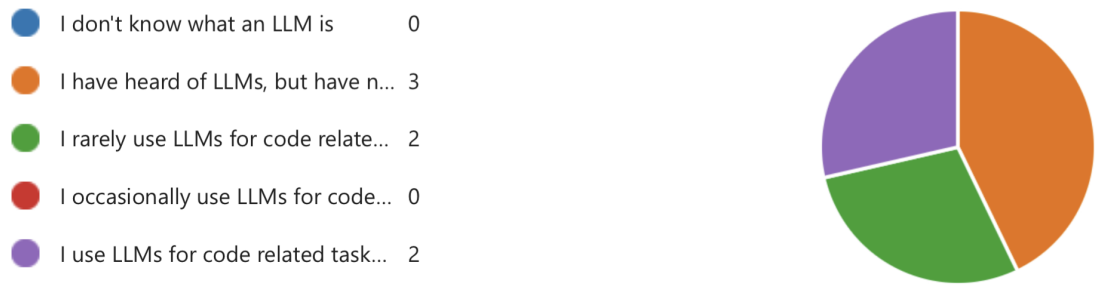


FIGURE 5.9: Participants' experience with LLMs for code related tasks

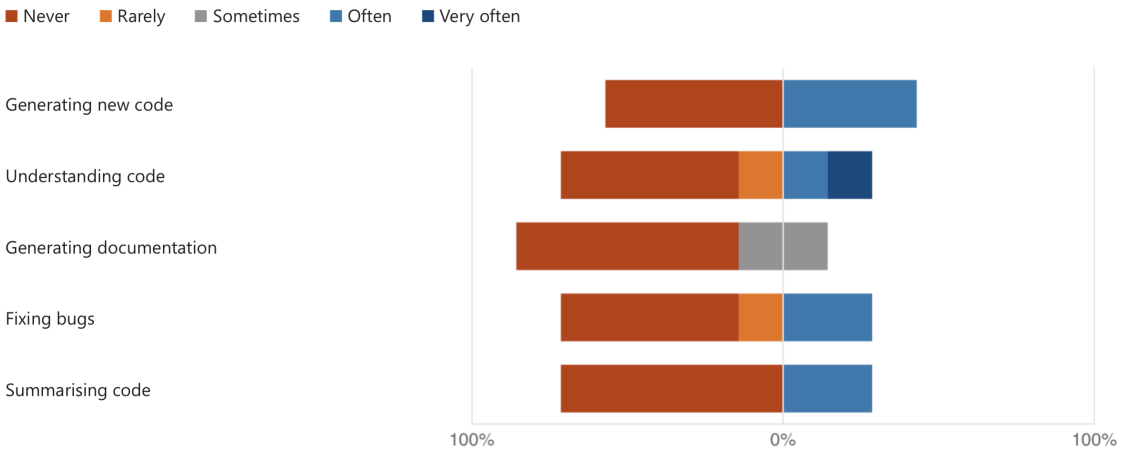


FIGURE 5.10: Participants' LLM usage for code related tasks

5.4.2 Discussion

As can be seen from table 5.11 in the previous section, all participants of the study onboard on new codebases very regularly, with one participant even claiming to onboard on a new codebase every month. Since there aren't many participants in this user study, this is important since these participants have a very good understanding of the onboarding process on a new codebase and can therefore accurately assess the usefulness of using an LLM for this purpose. From figure 5.8, it is also clear that the participants do not use any LLM tools yet to onboard on a new codebase. One explanation for this could be that the company where this experiment was done has strict rules regarding LLM usage with their source code. However, from figures 5.9 and 5.10, it's clear that over half of the participants have experience with using LLMs for code related tasks, and some of them even actively use them outside of their job.

5.4.3 Tasks

Table 5.12 contains data about each participant's answers for each task. This includes time to finalize an answer and if the answer was correct, together with an average to get an indication how long each task would take.

		Task 1		Task 2		Task 3		Task 4	
ID	Group	Time	Correct?	Time	Correct?	Time	Correct?	Time	Correct?
1	2	4:17	✓	7:09	✓	-	-	-	-
2	2	13:03	✓	10:57	✗	1:53	✓	15:00	✗
3	1	7:13	✓	11:03	✓	5:13	✓	13:58	✗
4	1	3:45	✓	6:11	✓	1:31	✓	9:32	✓
5	1	8:15	✓	7:51	✓	2:24	✓	11:23	✗
6	2	14:07	✓	6:20	✓	1:47	✓	8:15	✓
7	2	8:56	✓	5:11	✓	1:24	✓	9:21	✓
Average		8:30	-	7:48	-	2:22	-	11:14	-
Average (group 1)		6:24	-	8:21	-	3:02	-	11:37	-
Average (group 2)		10:05	-	7:24	-	1:41	-	10:52	-

TABLE 5.12: Task results

Figures 5.11 to 5.14 in the next few sections contain the results from the inter-task questions, which contain information about the participants' perceived difficulty of the tasks/code they looked at. The figures for tasks 3 and 4 have one less participant, since one participant didn't complete these tasks due to being an expert on that codebase. Because of the unique coincidence of one participant being an expert on the second codebase, it felt unnecessary to complete these tasks as they knew the exact fixes for the problems without even looking at the code. Instead, this participant was asked how using the tool could improve the onboarding process on that codebase, to which they answered "Using the tool is very promising when getting a quick view of what the code does, but the current model lacks the ability to understand the finer details".

As mentioned in section 4.6.1, for each task, the participants were also asked to explain the essence of the code they just looked at. Since the answers of these questions can not be

used verbatim, due to containing information about the codebase, a summary of correct answers is provided in the sections below.

Task 1

For this task, 6 out of 7 participants correctly identified the essence of the code. The last participant only mentioned "parsing integers from string", which is essentially what happens, but doesn't capture the full essence of the code (i.e. what is parsed/ for what reason). The person who didn't fully identify the essence of the code is part of the second group, who used the LLM tool for this task. Figure 5.11 shows the split results between the two groups. From this figure, it can be seen that group 1 (who didn't use the LLM tool for this task) overall had more trouble with understanding the task and found the task more mentally challenging than the second group who did use the LLM tool. However, group 1's participants were eventually also more confident in the answer they produced. Looking at the results in table 5.12, it can also be seen that the average completion time for this task was 6:24 for the group without the LLM tool, and 10:05 for the group with the LLM tool.

For task 1, there was one additional remark from a participant:

ID	Group	Remark
7	2	"LLM wasn't helpful, it kept hallucinating"

TABLE 5.13: Participants' remarks about task 1

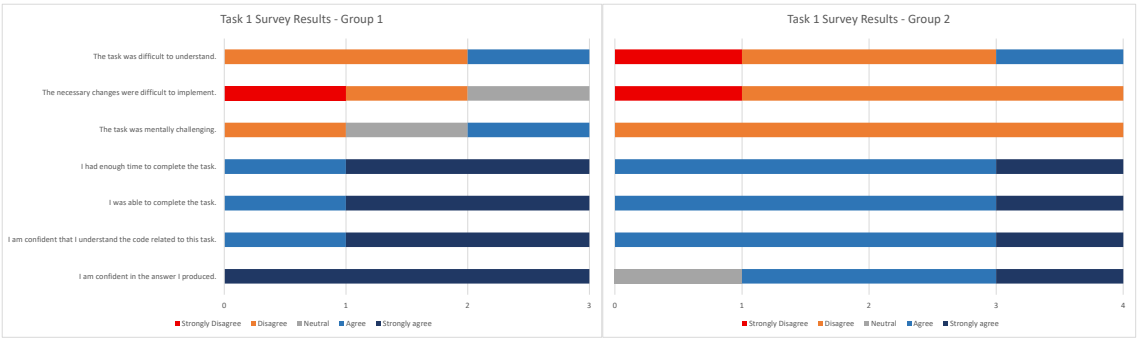


FIGURE 5.11: Inter-Task Survey Task 1 - Split between groups

Task 2

For the second task, all participants understood the essence of the code. However, three of the answers essentially mentioned "parsing http responses", which could have been explained in a little bit more detail. Looking at figure 5.12, the second group (using the LLM tool) reported less difficulties when understanding the task. The rate of understanding the source code is nearly equal, with 3/3 participants from group 1 reporting they understand the code, and 3/4 participants from the second group reporting the same. The average completion time for this task was 8:21 for the group without the LLM tool, and 7:24 for the group with the tool.

For the second task, three participants left additional remarks.

ID	Group	Remark
1	2	"Spot on answer by the model"
3	1	"Stack trace could speed up finding the issue and also how to fix it later"
7	2	"LLM provided correct answers"

TABLE 5.14: Participants' remarks about task 2

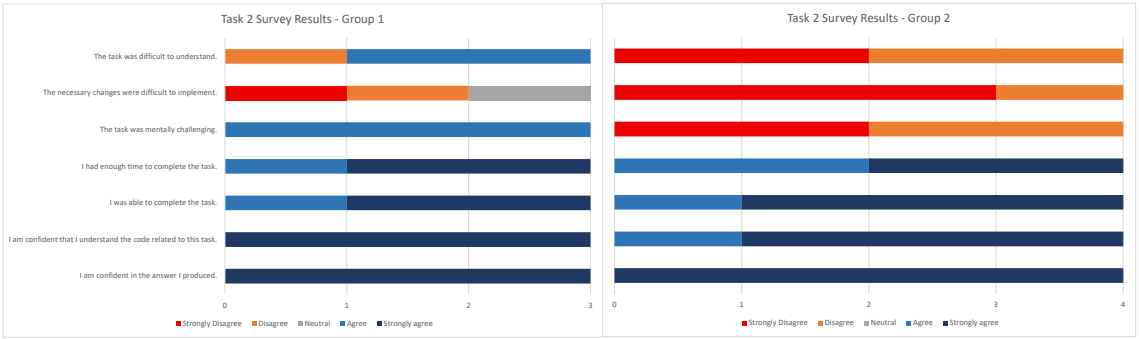


FIGURE 5.12: Inter-Task Survey Task 2 - Split between groups

Task 3

Task 3 had by far the fastest completion time, shown in table 5.12. Additionally, all participants understood the essence of the code and were able to provide correct answers. For this task, group 2 did not use the LLM tool, while the first group did. From the survey results in figure 5.13, it can also be seen that no participants had trouble understanding the task, and were confident in understanding the source code. Looking at the average completion times in table 5.12, it can be seen that group 1 completed this task in an average of 3:02, while group 2 did this in 1:41. For this task, there were no additional remarks from the participants.

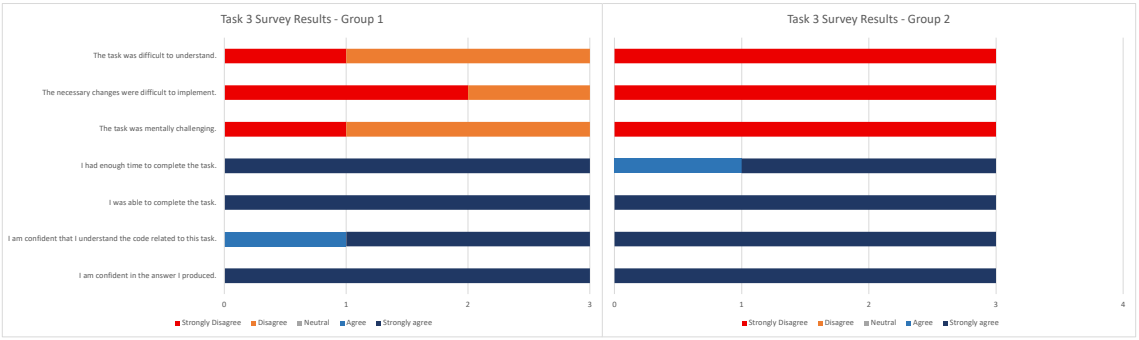


FIGURE 5.13: Inter-Task Survey Task 3 - Split between groups

Task 4

The last task, compared to the other tasks, took the longest to complete on average, with an average time of 11:14, as can be seen in figure 5.12. For this task, three participants didn't get the correct answer to the task, two of them using the LLM tool and the other not using it. Looking at figure 5.14, it can also be seen that participants had more trouble

understanding the task, and had lower confidence that they understood the code, compared to the other tasks. Group 1, the group that used the LLM tool for this task, also indicated they had more trouble understanding the task and had less confidence in their answers compared to the participants who used documentation instead. Looking at table 5.12, it can also be seen that participants using the LLM tool took longer to produce an answer than the participants without the LLM tool.

For this last task, there were two additional remarks from the participants:

ID	Group	Remark
3	1	"You could have next step that we include **[BUSINESS LOGIC]** and see how llm can find the answer with more relevant context"
4	1	"Business logic knowledge could give an advantage"

TABLE 5.15: Participants' remarks about task 4

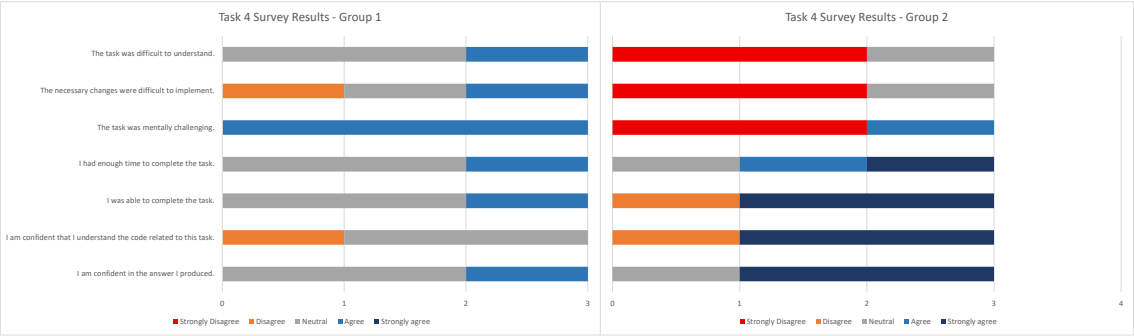


FIGURE 5.14: Inter-Task Survey Task 4 - Split between groups

5.4.4 Discussion

Table 5.12 showed the completion rates and times for each task. One important thing to note is that participant 1 did not perform tasks 3 and 4, because they coincidentally were an expert on the codebase and knew exactly how to complete the tasks, without even looking at the source code. These results show that most participants were able to complete the tasks in the given time, and were also able to give the correct answers. However, participants 2, 3, and 5 didn't manage to correctly answer all tasks. Looking at the time to produce a final answer, there is also quite a difference between the tasks, with task 4 taking the longest to complete, and task 3 taking the shortest. Looking at the inter-task survey results from the individual tasks, this difference in difficulty is also apparent. For task 3 (and task 2 as well), most participants are confident they understand the code. For task 4, the results are more mixed, with some participants claiming they understand the code, while others weren't so sure. This difference in difficulty might be seen as a threat to validity, since the participants who used the LLM tool for task 3 didn't get as much time with the tool as participants in study group 2. However, this difference in difficulties between tasks is an accurate representation of first tasks junior developers get when onboarding on a new codebase. This is further reinforced by the fact that these

tasks are actual tasks in the codebases, which have been selected together with two senior developers to accurately reflect first tasks for junior developers.

Further looking at the results in the previous section, it can be seen that using the LLM tool (on average) lead to more time spent on creating a final answer compared to not using the tool at all. For all tasks, except task 2, the participants using the LLM tool took longer to create an answer. For task 4, this might be explained by the fact that the LLM tool didn't have the required business logic in order to accurately answer the task. For tasks 1 and 3, this is likely caused by the participants figuring out how the tool works. From the figures showing the survey results split between the groups, it can be seen that the participants using the LLM tool for tasks 1 and 2 generally had an easier time understanding the tasks, and found the tasks to be less mentally challenging than their counterparts not using the LLM tool. However, these participants using the LLM were also less confident in the answers they produced. This lack of confidence might have been caused by the performance of the LLM, with one participant on the first task noting that the LLM kept hallucinating. Other participants didn't have the LLM hallucinate as much and were able to get useful answers from it. The LLM hallucinating is most likely caused by the size of the model used in the user study - Llama3 7b with 2k context size, which was chosen during the experiment in section 5.3 to be the best setup with the hardware limitations we had. These results imply that using an LLM tool helps remove some of the mental fatigue in finding information about the codebase, as an LLM can quickly produce this information. However, due to the unexplainable results an LLM produces, users of the tool are also more sceptical regarding the results it produces. One exception to this can be seen in the results for task 4. For this task, the participants who used the LLM tool had a harder time understanding the task and were less confident in their understanding of the code than the participants who didn't use the LLM tool. This discrepancy in the results was likely caused by the fact that participants needed business logic to fully understand the code, which only existed in the documentation for the codebase. The LLM tool did not have access to this documentation, so participants had a harder time coming up with the right answer for this task. Both participants who left remarks for task 4 also mentioned this lack of business logic when using the LLM tool. This lack of including relevant documentation is a clear limitation of the system, since the LLM did not have any knowledge about this and could therefore not help the user in understanding the piece of code.

5.4.5 Post-Task Survey

Figures 5.15 and 5.16 contain the summarized results of all questions asked about usage of the LLM compared to traditional onboarding methods. The last figure, 5.17, contains the participants' answers to the question "How valuable would the LLM tool be for your work if it could be added to your development environment?". Lastly, table 5.16 contains the participants' explanations for the answers shown in figure 5.17

5.4.6 Discussion

Looking at the results from the post-task survey in figures 5.15 and 5.16, most participants found that the LLM tool generally gave helpful answers, with two participants disagreeing with that statement. On the contrary, most participants also thought that the LLM gave them more confusing results than traditional comprehension methods, with one partici-

ID	Current State	Larger Scope	Explanation
1	Valuable	Very valuable	"I think LLM can definitely help explaining syntax and part of the code. This would speedup the learning curve for new developers."
2	Valuable	Very valuable	"LLM tool is good at answering initial questions when I have zero clue about the codebase, and quickly narrows down what I need to further know/do to solve the task."
3	Very valuable	Very valuable	"it is another helpful tool to comprehend the code and fix the bugs and also writing new new piece of code in shortest time ."
4	Valuable	Very valuable	"When not knowing where to start (looking), an LLM can quickly analyze and direct you to the right place. When it comes to actual implementation, traditional methods might still be superior due to their fast and direct/concrete nature."
5	Limited value	Very valuable	"1. Current model is hard to install & use 2. Current model is a bit slow and produces inaccurate results very often (which makes it untrustworthy)"
6	Valuable	Valuable	"LLM module can be useful to understand when you can large chain of methods calling and complex logic"
7	Average value	Valuable	"hallucinates a bit too much"

TABLE 5.16: Participants' explanations for their answers regarding the usefulness of LLM tools for code comprehension

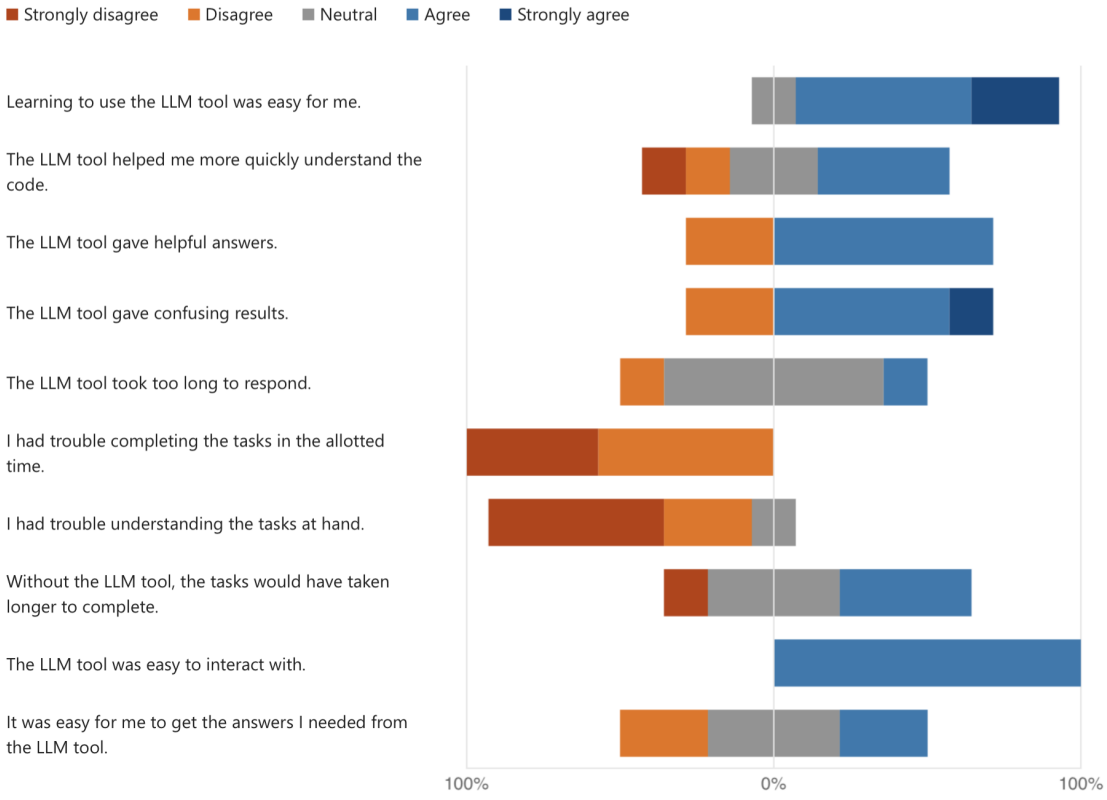


FIGURE 5.15: Post-Task Survey LLM Questions

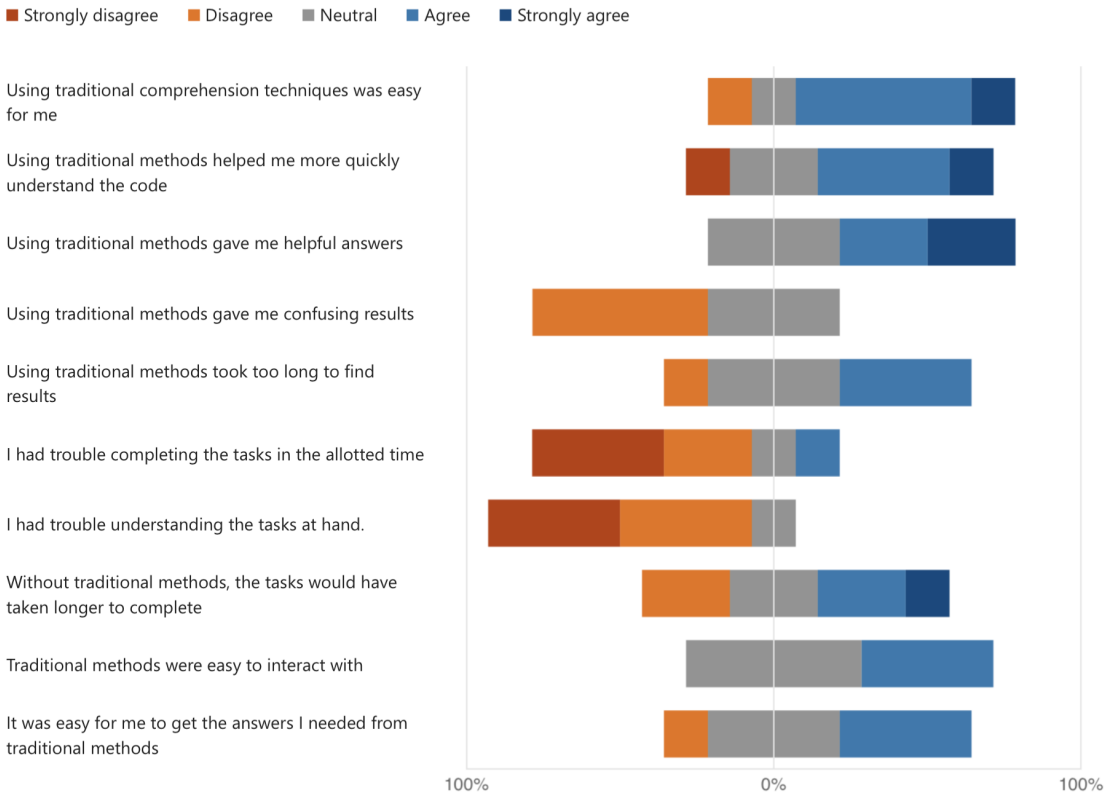


FIGURE 5.16: Post-Task Survey Non-LLM Questions

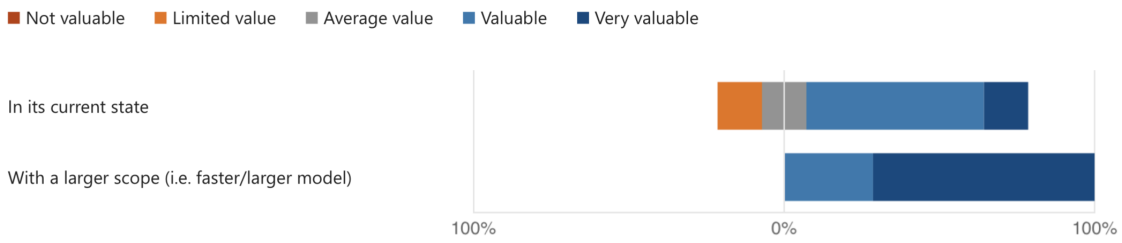


FIGURE 5.17: Participants' ratings regarding the usefulness of the LLM tool

pant specifically mentioning the fact that the LLM hallucinated too much. Again, with two other participants disagreeing with this statement. These contrasting results imply that the LLM is generally helpful when first trying to understand a codebase, but isn't great in answering questions about fine details. This inability to answer questions about fine details is likely caused by the size of the model used, as well as the amount of context used in the prompt. However, we are aware that larger models also struggle with their attention to details. These results also correspond to the explanations the participants gave in table 5.16, where participants indicated that the LLM tool is helpful when first trying to understand the codebase, but struggles with finer details. With this, RQ1 is answered.

Even though the LLM tool sometimes produced sub-par results, most participants found the tool easy to interact with and in most cases indicated that using the tool helped them complete the tasks faster than without the tool.

5.4.7 Logged Interactions

Table 5.17 shows the total amount of interactions each participant had with the LLM tool for the tasks they had access to the tool. The rest of this section contains all interactions with the LLM tool the participants had, excluding any sensitive information from the codebases they interacted with. Each question asked by participants is shown in italic, with a summary of the LLM's response in normal text below that.

ID	Task 1	Task 2	Task 3	Task 4
1	5	2	-	-
2	2	1	-	-
3	-	-	1	4
4	-	-	1	3
5	-	-	1	3
6	8	1	-	-
7	4	2	-	-

TABLE 5.17: Amount of LLM interactions per participant, per task

Participant 1

Task 1
What does this code do?

The LLM explains what the code does in detail.
Do you see a problem with this code?
The LLM mentions the code is well structured, and mentions no specific problems with the code.
*Entering the string **[input string]** gives a NumberFormatException, can you tell me why?*
The LLM hallucinates non-numeric characters in the input string.
[input string] does not have non numeric characters
The LLM apologizes for its mistake and says that the input should be able to be parsed (which it can't).
*Can you check if Integer.parseInt with **[input string]** works?*
The LLM doesn't understand the question and would like more information on "works".
Does it throw an exception or does it parse correctly?
The LLM correctly determines that the input string throws a NumberFormatException.

Task 2
What does the selected code do?
The LLM answers with a detailed explanation of the selected code, explaining a 204 HTTP code as well.
Using the response body when the result is 204 gives an error can you tell me why exception would occur
The LLM correctly identifies the problem, mentioning an empty response body for a 204 request and identifying the piece of code where this would lead to a problem

Participant 2

Task 1
What does the selected code do?
The LLM explains the essence of the code in detail, explaining relevant parts well.
what could cause NumebrFormatExeception with input string ̈26102023102010̈?
The LLM responds with an unrelated answer, talking about regex.

Task 2
What does the selected code do?
The LLM responds by explaining the code in detail, including error handling when receiving a 204 HTTP response.

Participant 3

Task 3
*Currently, users **[only get a limited number of results back]** This limit needs to be removed, so users **[get more results]***
The LLM correctly identified which parameter had to be removed in order to return more results.

Task 4
*Currently, **[certain results]** are excluded when making an API call to **[API]**. These have to be included in the response*
The LLM didn't have enough context to help the participate, and asked for more context
*Currently, **[certain results]** are excluded when making an API call to **[API]**. These have*

to be included in the response
This time, the LLM had too much context and the prompt leaked, not providing the participant with any useful information.
*Currently, **[certain results]** are excluded when making an API call to **[API]**. These have to be included in the response*
This time, the LLM gave a good response where it referenced the relevant piece of code which filtered out certain results.
*this is the filter how shoudl include **[certain results]***
The LLM didn't know how to include the **[certain results]**, since it has no knowledge of these.

Participant 4

Task 3
*The following method only returns **[a certain number of results]**. How can we change this?*
The LLM correctly identified the parameter which needed to be changed, and suggested a change for it.

Task 4
*Which method is responsible for filtering **[certain results]**?*
The LLM correctly identified which method is responsible for filtering out certain results.
Does this method exclude some results?
The LLM responds by confirming the participant's suspicions about filtering out results
*How can we include **[certain results]** given the filtering in this method?*
The LLM responds by giving a suggestion how to include the **certain results** in a response.

Participant 5

Task 3
What does the selected code do?
The LLM responds by hallucinating and leaking the prompt, not helping the participant.

Task 4
What does the selected code do?
The LLM explains what the code does, mentioning the types of results returned.
*Does it return both **[certain results]** and **[other results]**?*
The LLM didn't have enough information to answer the participant's question.
*In which part are **[certain results]** returned?*
The LLM gives a wrong answer, and can't help the participant.

Participant 6

Task 1
What does the selected code do?
The LLM explains the essence of the code correctly.
How can I use the selected code?
The LLM explains how the selected code can be used to parse a certain input, and gives example inputs.
What kind of exception can this code throw?

The LLM explains the types of exceptions that the selected piece of code can throw.
Where number format exception can be thrown?
The LLM leaks a bit of the prompt and doesn't help the participant.
If I pass string value what will be the exception?
The LLM correctly identifies problems that passing a string to `Integer.ParseInt` can give.
*What is I pass **[certain input]**?*
The LLM hallucinates white spaces in the given input string and misidentifies the error.
*Why does the input **[given input]** give an exception?*
The LLM hallucinates an exclamation mark in the input string, but does correctly identify where the error comes from.

Task 2
What could be the potential issue in this code?
The LLM identifies a few possible issues in the code, mostly related to code style. It does not help the participant solve the given problem.

Participant 7

Task 1
What does the selected code do?
The LLM responds by explaining what the code does in detail.
*What happens if I try to set the **[input string as input to the function]**?*
The LLM hallucinates hyphens in the input string, and misidentifies the problem.
*Does **[given input string match expected input]**?*
The LLM is still hallucinating with the hyphens.
*Compare **[input string]** to max int*
The LLM mentions the maximum size of an integer, which the participant compares to the given input string and determines it is too large.

Task 2
Explain http code 204
The LLM explains about HTTP code 204, and explains how this HTTP code is handled in the selected code as well.
What happens in this piece of code if the response body is empty?
The LLM says that the code will throw an error if the response body is empty, but does not correctly identify the exact error that will be thrown.

5.4.8 Discussion

Looking at the results in the previous section it's clear that some participants had to interact more with the LLM tool than others, constantly refining their questions, in order to get the desired results. For example, participant 1 (who doesn't use LLM tools), asked 5 separate questions in order to get an answer for task 1. Compare that to participant 3 (who does use LLM tools often), who only asked 1 question for task 3 and 2 separate questions for task 4 (excluding the repeated questions due to the prompt leaking). From this, we can see that users who have a better understanding of how to formulate questions for an LLM get the answers they're looking for faster than users who don't have this knowledge. Looking at all interactions, it can be seen that most developers start their interaction with asking very broad questions like "What does the selected code do?" or "Which method is

responsible for *[certain task]?*", after which more in depth questions related to their task are asked. These findings about developers' interaction with the LLM tool answer RQ2.

Again looking at the participants' interactions with the LLM tool, there are a few cases where the LLM misinterprets the participant's intention. For example when participants asked the LLM about the code's potential issues (participant 1 task 1 & participant 6 task 2), to which the LLM responded with answers related to code style. This misinterpretation of a participant's written prompt hinders the participant's ability to answer the tasks quickly, and could lead to the person getting annoyed with the LLM instead of thinking the responses are helpful.

Chapter 6

Conclusion

This research aimed to investigate the usefulness of using an LLM-based tool to help developers understand undocumented pieces of code, as opposed to using more traditional code comprehension methods. This section will summarize the findings discussed in section 5, and will answer the following research questions:

- RQ1: To what extent can an LLM-based tool help a developer understand a large codebase?
- RQ2: How do professional developers interact with an LLM-based tool in order to understand a piece of code?

The rest of this section will summarize other conclusions found during the research, as well as explore opportunities for future work directions.

6.1 Research Questions

6.1.1 RQ1: extent of understanding code using an LLM-based tool

As can be seen from the results of the user study in section 5.4, most participants reported that they feel an LLM tool, such as the one created during this research, can be very valuable when getting a quick overview of a codebase. Participants reported that the LLM tool created in this research sometimes hallucinated and couldn't comprehend the finer details of the code. As mentioned in section 2.2.4, this is mainly caused by the lack of context and the smaller models used during this research, due to running the models locally on participants' laptops, because of limitations from the company this research was performed at. However, despite the limited scope of the tool used, the participants were able to complete most tasks in the user study. One area where the LLM tool was ineffective when solving a task was for the fourth task, where it lacked the external business logic to form a correct answer. To answer RQ1: developers can use LLM based tools to get a low-level understanding of how the code works, but will struggle with gaining a higher level understanding of the purpose of the code without using external documentation.

6.1.2 RQ2: developers' interactions with LLM tools

As shown in sections 5.4.7 and 5.4.8, there is a difference in how different participants interacted with the LLM tool. Participants who had more experience working with LLM tools generally constructed more direct questions about the tasks they were trying to solve compared to the broader questions created by the participants with little to no experience using LLM tools. So to answer the RQ: most developers start their interaction with the LLM tool by asking broad questions about the code, before narrowing down on specific parts of the code they want to know more about. However, results have also shown that this is a very subjective topic, with more experienced LLM users constructing more detailed questions.

6.2 Other conclusions

6.2.1 Subjectiveness of interpreting LLM answers

Looking at the results from both the experiment (section 5.3) and the user study (section 5.4), it becomes clear that the interpretation of LLM responses is very subjective. In some

cases, people liked the responses the LLM gave even though they weren't completely correct, and in some others this was disliked, as seen in section 5.3.7. Similarly, subjectiveness was also a part of classifying the LLM's bad responses. As mentioned in section 5.3.10, there is a fine line between an LLM response being incorrect and being a hallucination, with the raters gravitating more towards it being incorrect.

6.3 Future Work

Larger Models. Due to the hardware limitations in this research, one obvious direction for future work would be to use larger LLM models with larger context sizes. In this research, when participants of the user study selected too much context to include in the prompt, the LLM prompt would be too large and leak the prompt or start hallucinating. Since the results with such a relatively small model are already promising, future research using larger models will be valuable in creating better tools for developers to aid them in comprehending code.

Additional context. The LLM tool used in this research used three additional types of context: the repository structure, the call hierarchy of a selected method, and pieces of similar code. This lead to the LLM being good at explaining the code on a technical level, but lacked the additional context to really put the code into perspective. This lack of external documentation was abundantly clear from the results of the fourth task, discussed in section 5.4.4, where the LLM lacked the necessary business logic to help the participants and hallucinated a lot when answering. For future work, adding extra context such as external documentation would likely lead to an increased performance by the LLM in explaining the higher-level purpose of the code. However, the amount of context should also be taken into account. From the results in section 5.3.9, it can be seen that having too much information in the prompt leads to the LLM giving incorrect answers and prompt leakage.

Additional validation. Since this research only included a set of 7 user study participants, logical future work would be to increase the amount of participants and further research the effects of using LLM-based tools to comprehend code. Additionally, future research should focus on increasing the amount of expert raters to validate LLM responses. Having more raters rate different types of context would lead to an increased LLM performance.

Promptless interaction. As mentioned in 5.4.7, the LLM sometimes misinterprets a user's input question and gives irrelevant answers. Future work in this direction should investigate a prompt-less approach, where users don't have to construct their own questions but can use pre-generated questions to find their answers. Alternatively, future research could look into modifying the questions a user puts into the prompt before sending them to the LLM (i.e. sentiment analysis). This way, the question could be reformulated in a way which the LLM can handle better, and even reducing the amount of hallucinations, as mentioned in section 2.2.4.

Context impact. In order to produce better results, research should be done into how different tyes of context (or lack thereof) impacts the LLM's ability to generate responses. Our research briefly explored this in section 5.3.10, but future work should further explore this by looking at how different contexts/ prompts lead to different LLM mistakes.

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Ekansh Agrawal, Omair Alam, Chetan Goenka, Medha Iyer, Isabela Moise, Ashish Pandian, and Bren Paul. Code compass: A study on the challenges of navigating unfamiliar codebases. *arXiv preprint arXiv:2405.06271*, 2024.

[3] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. *arXiv preprint arXiv:2005.00653*, 2020.

[4] Toufique Ahmed and Premkumar Devanbu. Few-shot training llms for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, 2022.

[5] Toufique Ahmed, Kunal Suresh Pai, Premkumar Devanbu, and Earl T Barr. Improving few-shot prompts with relevant static analysis products. *arXiv preprint arXiv:2304.06815*, 2023.

[6] Meta AI. Introducing meta llama 3: The most capable openly available llm to date. https://ai.meta.com/blog/meta-llama-3/, April 2024. Accessed: 30-05-2024.

[7] Nomic AI. Introducing nomic embed: A truly open embedding model. https://blog.nomic.ai/posts/nomic-embed-text-v1, February 2024. Accessed: 10-06-2024.

[8] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code, 2019. arXiv:1808.01400.

[9] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[10] Oliver Arafat and Dirk Riehle. The commenting practice of open source. pages 857–864, 10 2009. doi:10.1145/1639950.1640047.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[12] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

[13] Alec Berntson. Azure ai search: Outperforming vector search with hybrid retrieval and ranking capabilities. https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/azure-ai-search-outperforming-vector-search-with-hybrid/ba-p/3929167, September 2023. Accessed: 21-06-2024.

[14] Scott Blinman and Andy Cockburn. Program comprehension: investigating the effects of naming style and documentation. In *AUIC*, pages 73–78, 2005.

[15] Krzysztof Borowski, Bartosz Balis, and Tomasz Orzechowski. Semantic code graph–an information model to facilitate software comprehension. *IEEE Access*, 2024.

[16] Ruven Brooks. Using a behavioral theory of program comprehension in software engineering. In *Proceedings of the 3rd international conference on Software engineering*, pages 196–201, 1978.

[17] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[18] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[20] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcodebert: Pre-training code representations with data flow, 2021. arXiv:2009.08366.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *Proceedings of the 26th conference on program comprehension*, pages 200–210, 2018.

[23] W John Hutchins. The georgetown-ibm experiment demonstrated in january 1954. In *Conference of the Association for Machine Translation in the Americas*, pages 102–114. Springer, 2004.

[24] Alekseĭ Ivakhnenko. Cybernetics and forecasting techniques.

[25] Ziwei Ji, Zihan Liu, Nayeon Lee, Tiezheng Yu, Bryan Wilie, Min Zeng, and Pascale Fung. RHO: Reducing hallucination in open-domain dialogues with knowledge grounding. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4504–4522, Toronto, Canada, July 2023. Association for Computational Linguistics. URL: https://aclanthology.org/2023.findings-acl.275, doi:10.18653/v1/2023.findings-acl.275.

[26] Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating hallucination in large language models via self-reflection. *arXiv preprint arXiv:2310.06271*, 2023.

[27] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[28] Junaed Younus Khan and Gias Uddin. Automatic code documentation generation using gpt-3. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–6, 2022.

[29] Amy J Ko, Brad A Myers, Michael J Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on software engineering*, 32(12):971–987, 2006.

[30] Jan Kocoń, Igor Cichecki, Oliwier Kaszyca, Mateusz Kochanek, Dominika Szydło, Joanna Baran, Julita Bielaniewicz, Marcin Gruza, Arkadiusz Janz, Kamil Kanclerz, et al. Chatgpt: Jack of all trades, master of none. *Information Fusion*, 99:101861, 2023.

[31] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. arXiv:2205.11916.

[32] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering*, 3:303–318, 2007.

[33] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. Comparing code explanations created by students and large language models. *arXiv preprint arXiv:2304.03938*, 2023.

[34] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[35] Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970.

[36] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[37] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 931–937, 2023.

[38] Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.

[39] Risto Miikkulainen and Michael G Dyer. Natural language processing with modular pdp networks and distributed lexicon. *Cognitive Science*, 15(3):343–399, 1991.

[40] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[41] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[42] Roberto Minelli, Andrea Mocci, and Michele Lanza. I know what you did last summer-an investigation of how developers spend their time. In *2015 IEEE 23rd international conference on program comprehension*, pages 25–35. IEEE, 2015.

[43] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding, 2024. arXiv:2307.08177.

[44] Noor Nashid, Mifta Sintaha, and Ali Mesbah. Retrieval-based prompt selection for code-related few-shot learning. In *Proceedings of the 45th International Conference on Software Engineering (ICSE'23)*, 2023.

[45] Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder, 2024. arXiv:2402.01613.

[46] Dr. Deb Bharadwaj Ofer Mendelevitch, Haihao Shen. Do smaller models hallucinate more? https://vectara.com/blog/do-smaller-models-hallucinate-more/, April 2024. Accessed: 19-07-2024.

[47] Delano Oliveira, Reydne Bruno, Fernanda Madeiral, and Fernando Castor. Evaluating code readability and legibility: An examination of human-centric studies. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 348–359. IEEE, 2020.

[48] Zoltán Porkoláb, Tibor Brunner, Dániel Krupp, and Márton Csordás. Codecompass: an open software comprehension framework for industrial usage. In *Proceedings of the 26th Conference on Program Comprehension*, pages 361–369, 2018.

[49] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[50] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[51] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[52] Paige Rodeghero, Cheng Liu, Paul W McBurney, and Collin McMillan. An eye-tracking study of java programmers and application to source code summarization. *IEEE Transactions on Software Engineering*, 41(11):1038–1054, 2015.

[53] Paige Rodeghero, Collin McMillan, Paul W McBurney, Nigel Bosch, and Sidney D'Mello. Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th international conference on Software engineering*, pages 390–401, 2014.

[54] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[55] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. arXiv:1907.10641.

[56] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 27–43, 2022.

[57] Maximilian Schreiner. Gpt-4 architecture, datasets, costs and more leaked. https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/, July 2023. Accessed: 31-05-2024.

[58] Elliot Soloway and Kate Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on software engineering*, (5):595–609, 1984.

[59] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the 25th IEEE/ACM international conference on Automated software engineering*, pages 43–52, 2010.

[60] M.-A. Storey. Theories, methods and tools in program comprehension: past, present and future. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 181–191, 2005. doi:10.1109/WPC.2005.38.

[61] Weisong Sun, Chunrong Fang, Yudu You, Yun Miao, Yi Liu, Yuekang Li, Gelei Deng, Shenghan Huang, Yuchen Chen, Quanjun Zhang, et al. Automatic code summarization via chatgpt: How far are we? *arXiv preprint arXiv:2305.12865*, 2023.

[62] Barbee E Teasley. The effects of naming style and expertise on program comprehension. *International Journal of Human-Computer Studies*, 40(5):757–770, 1994.

[63] Ted Tenny. Program readability: Procedures versus comments. *IEEE Transactions on Software Engineering*, 14(9):1271–1279, 1988.

[64] Scott R Tilley and Dennis B Smith. Coming attractions in program understanding. 1996.

[65] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[67] Anneliese Von Mayrhauser and A Marie Vans. From program comprehension to tool requirements for an industrial environment. In *[1993] IEEE Second Workshop on Program Comprehension*, pages 78–86. IEEE, 1993.

[68] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[69] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[70] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System Modeling and Optimization: Proceedings of the 10th IFIP Conference New York City, USA, August 31–September 4, 1981*, pages 762–770. Springer, 2005.

[71] J Wiegand et al. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2):371–383, 2004.

[72] Hongqiu Wu, Hai Zhao, and Min Zhang. Code summarization with structure-induced transformer. *arXiv preprint arXiv:2012.14710*, 2020.

[73] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E Hassan, and Shanping Li. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10):951–976, 2017.

[74] Jia-Yu Yao, Kun-Peng Ning, Zhen-Hui Liu, Mu-Nan Ning, and Li Yuan. Llm lies: Hallucinations are not bugs, but features as adversarial examples. *arXiv preprint arXiv:2310.01469*, 2023.

[75] Yu Zhou, Xin Yan, Wenhua Yang, Taolue Chen, and Zhiqiu Huang. Augmenting java method comments generation with context information based on neural networks. *Journal of Systems and Software*, 156:328–340, 2019.

Appendix A

Call Hierarchy Representations

Automated Evaluation

Representation	Vector distance	Token amount
YAML 1	0,31997	627
YAML 2	0,33098	627
YAML 3	0,27682	627
YAML 4	0,33823	627
YAML 5	0,33823	627
Nested list 1	0,36783	930
Nested list 2	0,31569	930
Nested list 3	0,32218	930
Nested list 4	0,41269	930
Nested list 5	0,32218	930
Grammar 1	0,36323	1334
Grammar 2	0,36323	1334
Grammar 3	0,36323	1334
Grammar 4	0,36323	1334
Grammar 5	0,36323	1334
Grammar list 1	0,23675	509
Grammar list 2	0,23675	509
Grammar list 3	0,24410	509
Grammar list 4	0,23675	509
Grammar list 5	0,23675	509
YAML avg	0,32085	627
Nested list avg	0,34811	930
Grammar avg	0,36323	1334
Grammar list avg	0,23822	509

Table A.1: Automated assessment of different call hierarchy representations

All LLM responses

YAML

Prompt size: 1661 tokens

Response 1

```
Based on the context provided, the following methods call the `
    withTemplate` method of `MetadataRow` class directly:

1. `buildFrom(this.statement, new MetadataResultSet(rowTemplate).
   setRows(attributesRows))` in `TypeMetadataResultSetBuilder.
   java`
2. `new MetadataResultSet(rowTemplate)` in `MetadataResultSet.
   java`

These methods are called during the process of building various
   types of metadata result sets.
```

Response 2

```
Based on the context provided, the following methods call the `
    withTemplate` method of `MetadataRow` class directly:

1. `buildFrom(this.statement, new MetadataResultSet(rowTemplate).
   setRows(attributesRows))` in `TypeMetadataResultSetBuilder.
   java`
2. `new MetadataResultSet(rowTemplate)` in `MetadataResultSet.
   java`

These methods are called during the process of building various
   types of metadata result sets, such as `TypeMetadataResultSet`
    and `MetadataResultSet`.
```

Response 3

```
Based on the context provided, the following methods call the `
    withTemplate` method of `MetadataRow` class directly:

1. `buildCatalogs()` in `CassandraMetadataResultSet`
2. `accept(ColumnMetadata)` in various test cases (`
   testIngIssue10`, `testIssue40`, `testIssue78`) and in the
   production code (`getColumns(String, String, String, String)`)
3. `buildTableTypes()` in `CassandraMetadataResultSet`
4. `accept(Entry<CqlIdentifier, UserDefinedType>)` in `
   getAttributes(String, String, String, String)`
5. `accept(FunctionSignature, FunctionMetadata)` in `getFunctions
   (String, String, String)`
```

```
These methods are responsible for building different types of
    metadata result sets and setting their rows using the provided
     `MetadataRow` instances.
```

Response 4

```
 Based on the context provided, the following methods call the `
    withTemplate` method of `MetadataRow` class directly:

1. `buildFrom(this.statement, new MetadataResultSet(rowTemplate).
   setRows(attributesRows))` in `TypeMetadataResultSetBuilder.
   java`
2. `new MetadataResultSet(rowTemplate)` in `MetadataResultSet.
   java`

These methods are called during the process of building various
    types of metadata result sets, such as `TypeMetadataResultSet
    `, `FunctionMetadataResultSet`, etc.
```

Response 5

```
 Based on the context provided, the following methods call the `
    withTemplate` method of `MetadataRow` class directly:

1. `buildFrom(this.statement, new MetadataResultSet(rowTemplate).
   setRows(attributesRows))` in `TypeMetadataResultSetBuilder.
   java`
2. `new MetadataResultSet(rowTemplate)` in `MetadataResultSet.
   java`

These methods are called during the process of building various
    types of metadata result sets, such as `TypeMetadataResultSet
    `, `FunctionMetadataResultSet`, etc.
```

Nested list

Response 1

```
 Based on the context provided, the following methods call the `
    withTemplate` method of the `MetadataResultSetBuilder` class
    directly:

1. `givenStatement_whenBuildCatalogs_returnExpectedResultSet()`
   in `MetadataResultSetsUnitTest.java`
2. `getSchemas(String, String)` in `CassandraDatabaseMetaData.
   java`
3. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.
   java`
```

4. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
5. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
6. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
7. `givenStatement_whenBuildUDTs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
8. `givenStatement_whenBuildFunctions_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`

These methods are unit tests that build expected result sets for various metadata queries using the given statement and the `MetadataResultSetBuilder`.

Response 2

Based on the context provided, the following methods call the `withTemplate` method of `MetadataResultSetBuilder` directly:

1. `givenStatement_whenBuildCatalogs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
2. `getSchemas(String, String)` in `CassandraDatabaseMetaData.java`
3. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
4. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
5. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
6. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
7. `givenStatement_whenBuildUDTs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
8. `givenStatement_whenBuildFunctions_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`

These methods are unit tests, and they use the `withTemplate` method to build result sets with specific templates for testing purposes.

Response 3

Based on the context provided, the following methods call the `withTemplate` method of `MetadataResultSetBuilder` directly:

1. `givenStatement_whenBuildCatalogs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
2. `getSchemas(String, String)` in `CassandraDatabaseMetaData.java`

3. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
4. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
5. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
6. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
7. `givenStatement_whenBuildUDTs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
8. `givenStatement_whenBuildFunctions_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`

These methods are unit tests, and they use the `withTemplate` method to build result sets for testing purposes.

Response 4

Based on the context provided, the following methods call the `withTemplate` method of `MetadataResultSetBuilder` directly:

1. `givenStatement_whenBuildCatalogs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
2. `getSchemas(String, String)` in `CassandraDatabaseMetaData.java`
3. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
4. `givenStatement_whenBuildUDTsWithNonJavaObjectTypes_returnEmptyResultSet()`, `givenStatement_whenBuildUDTsNotSpecifyingSchemaPattern_returnExpectedResult()`, `givenStatement_whenBuildUDTs_returnExpectedResultSet()`, and `givenStatement_whenBuildUDTsNotSpecifyingTypeNamePattern_returnExpectedResu()` in `MetadataResultSetsUnitTest.java`
5. `getUDTs(String, String, String, int[])` in `CassandraDatabaseMetaData.java`
6. `accept(TableMetadata)` in `TypeMetadataResultSetBuilder.java`
7. `givenStatement_whenBuildAttributes_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
8. `getColumns(String, String, String, String)` in `CassandraDatabaseMetaData.java`
9. `testIngIssue10()`, `testIssue40()`, `testIssue40()`, and `testIssue78()` in `JdbcRegressionUnitTest.java`
10. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
11. `givenStatement_whenBuildIndexes_returnExpectedResultSet()` and `givenStatement_whenBuildIndexes_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
12. `getFunctions(String, String, String)` in `CassandraDatabaseMetaData.java`

13. `givenStatement_whenBuildFunctions_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`

These methods are the ones that directly call the `withTemplate` method of `MetadataResultSetBuilder`. However, it's important to note that there might be other indirect ways these methods get called through other methods or classes.

Response 5

Based on the context provided, the following methods call the `withTemplate` method of `MetadataResultSetBuilder` directly:

1. `givenStatement_whenBuildCatalogs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
2. `getSchemas(String, String)` in `CassandraDatabaseMetaData.java`
3. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
4. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
5. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
6. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
7. `givenStatement_whenBuildUDTs_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`
8. `givenStatement_whenBuildFunctions_returnExpectedResultSet()` in `MetadataResultSetsUnitTest.java`

These methods are unit tests, and they use the `withTemplate` method to build result sets for testing purposes.

Grammar

Response 1

The methods that call the selected method "getPrimaryKeys(String, String, String)" directly are:

1. getPrimaryKeys(String, String, String) in CassandraDatabaseMetaData.java
2. testIssue77(), testIngIssue10(), testIssue40(), and testIssue78() in JdbcRegressionUnitTest.java

These methods are the ones that invoke the "getPrimaryKeys(String, String, String)" method directly without any further method calls in between.

Response 2

```
 The methods that call the selected method "getPrimaryKeys(String
     , String, String)" directly are:

1. getPrimaryKeys(String, String, String) in
   CassandraDatabaseMetaData.java
2. testIssue77(), testIngIssue10(), testIssue40(), and
   testIssue78() in JdbcRegressionUnitTest.java

These methods are the ones that invoke the "getPrimaryKeys(String
    , String, String)" method directly without any further method
    calls in between.
```

Response 3

```
 The methods that call the selected method "getPrimaryKeys(String
     , String, String)" directly are:

1. getPrimaryKeys(String, String, String) in
   CassandraDatabaseMetaData.java
2. testIssue77(), testIngIssue10(), testIssue40(), and
   testIssue78() in JdbcRegressionUnitTest.java

These methods are the ones that invoke the "getPrimaryKeys(String
    , String, String)" method directly without any further method
    calls in between.
```

Response 4

```
 The methods that call the selected method "getPrimaryKeys(String
     , String, String)" directly are:

1. getPrimaryKeys(String, String, String) in
   CassandraDatabaseMetaData.java
2. testIssue77(), testIngIssue10(), testIssue40(), and
   testIssue78() in JdbcRegressionUnitTest.java

These methods are the ones that invoke the "getPrimaryKeys(String
    , String, String)" method directly without any further method
    calls in between.
```

Response 5

```
 The methods that call the selected method "getPrimaryKeys(String
     , String, String)" directly are:
```

1. getPrimaryKeys(String, String, String) in CassandraDatabaseMetaData.java
2. testIssue77(), testIngIssue10(), testIssue40(), and testIssue78() in JdbcRegressionUnitTest.java

These methods are the ones that invoke the "getPrimaryKeys(String, String, String)" method directly without any further method calls in between.

Grammar with lists

Response 1

The methods that call the `withTemplate` method directly are:

1. `buildCatalogs()` in `CassandraDatabaseMetaData.java`
2. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
3. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
4. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
5. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
6. `accept(Entry<CqlIdentifier, UserDefinedType>)` in `TypeMetadataResultSetBuilder.java`
7. `accept(TableMetadata)` in `TypeMetadataResultSetBuilder.java`
8. `accept(FunctionSignature, FunctionMetadata)` in `FunctionMetadataResultSetBuilder.java`

These methods are called from various test cases in the `MetadataResultSetsUnitTest.java` file and from the `getCatalogs()` method in `CassandraDatabaseMetaData.java`.

Response 2

The methods that call the `withTemplate` method directly are:

1. `buildCatalogs()` in `CassandraDatabaseMetaData.java`
2. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
3. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
4. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
5. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
6. `accept(Entry<CqlIdentifier, UserDefinedType>)` in `TypeMetadataResultSetBuilder.java`
7. `accept(TableMetadata)` in `TypeMetadataResultSetBuilder.java`

8. `accept(FunctionSignature, FunctionMetadata)` in `FunctionMetadataResultSetBuilder.java`

These methods are called from various test cases in the `MetadataResultSetsUnitTest.java` file and from the `getCatalogs()` method in `CassandraDatabaseMetaData.java`.

Response 3

The methods that call the `withTemplate` method directly are `buildCatalogs()`, `accept(KeyspaceMetadata)`, `accept(TableMetadata)`, `accept(ColumnMetadata)`, and `buildTableTypes()`. These methods are called in various contexts, such as building catalogs, schemas, tables, types, and columns. The call hierarchy shows that these methods are ultimately used in unit tests to verify the expected results of metadata queries.

For example, `buildCatalogs()` is called in a test method `givenStatement_whenBuildCatalogs_returnExpectedResultSet()` in the file `MetadataResultSetsUnitTest.java`. Similarly, `accept(KeyspaceMetadata)` and `accept(TableMetadata)` are called in various test methods to build schemas, keyspaces, and other metadata structures.

Therefore, the selected method `withTemplate` is an essential part of building and querying metadata in your Cassandra JDBC wrapper application. It is used to populate a new `MetadataRow` object based on a given template and an array of values. This functionality is crucial for constructing various types of metadata result sets, such as those for columns, user-defined types (UDTs), functions, etc.

Response 4

The methods that call the `withTemplate` method directly are:

1. `buildCatalogs()` in `CassandraDatabaseMetaData.java`
2. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
3. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
4. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
5. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
6. `accept(Entry<CqlIdentifier, UserDefinedType>)` in `TypeMetadataResultSetBuilder.java`
7. `accept(TableMetadata)` in `TypeMetadataResultSetBuilder.java`

8. `accept(FunctionSignature, FunctionMetadata)` in `FunctionMetadataResultSetBuilder.java`

These methods are called from various test cases in the `MetadataResultSetsUnitTest.java` file and from the `getCatalogs()` method in `CassandraDatabaseMetaData.java`.

Response 5

The methods that call the `withTemplate` method directly are:

1. `buildCatalogs()` in `CassandraDatabaseMetaData.java`
2. `accept(KeyspaceMetadata)` in `SchemaMetadataResultSetBuilder.java`
3. `accept(TableMetadata)` in `TableMetadataResultSetBuilder.java`
4. `accept(ColumnMetadata)` in `ColumnMetadataResultSetBuilder.java`
5. `buildTableTypes()` in `TableMetadataResultSetBuilder.java`
6. `accept(Entry<CqlIdentifier, UserDefinedType>)` in `TypeMetadataResultSetBuilder.java`
7. `accept(TableMetadata)` in `TypeMetadataResultSetBuilder.java`
8. `accept(FunctionSignature, FunctionMetadata)` in `FunctionMetadataResultSetBuilder.java`

These methods are called from various test cases in the `MetadataResultSetsUnitTest.java` file and from the `getCatalogs()` method in `CassandraDatabaseMetaData.java`.

Appendix B

Pre-Experiment Parameters

ID	Graph ID	Context Size	Similar Snippets	Call Hierarchy	Repository Structure	Use Filenames	Test Directory	Plaintext explanation
1	1	1024	TRUE	TRUE	TRUE	TRUE	TRUE	All context, repository structure with file names and test directory
2	2	1024	TRUE	TRUE	TRUE	TRUE	FALSE	All context, repository structure with file names, without test directory
3	3	1024	TRUE	TRUE	TRUE	FALSE	TRUE	All context, repository structure without file names, with test directory
4	4	1024	TRUE	TRUE	TRUE	FALSE	FALSE	All context, repository structure without file names and test directory
5	5	1024	TRUE	FALSE	FALSE	-	-	Only similar code snippets
6	6	1024	FALSE	TRUE	FALSE	-	-	Only call hierarchy
7	7	1024	FALSE	FALSE	TRUE	TRUE	TRUE	Only repository structure, with file names and test directory
8	8	1024	FALSE	FALSE	TRUE	TRUE	FALSE	Only repository structure, with file names, without test directory
9	9	1024	FALSE	FALSE	TRUE	FALSE	TRUE	Only repository structure, without file names, with test directory
10	10	1024	FALSE	FALSE	TRUE	FALSE	FALSE	Only repository structure, with file names, without test directory
11	11	1024	FALSE	FALSE	FALSE	-	-	No context
12	13	2048	TRUE	TRUE	TRUE	TRUE	TRUE	All context, repository structure with file names and test directory
13	14	2048	TRUE	TRUE	TRUE	TRUE	FALSE	All context, repository structure with file names, without test directory
14	15	2048	TRUE	TRUE	TRUE	FALSE	TRUE	All context, repository structure without file names, with test directory
15	16	2048	TRUE	TRUE	TRUE	FALSE	FALSE	All context, repository structure without file names and test directory
16	17	2048	TRUE	FALSE	FALSE	-	-	Only similar code snippets
17	18	2048	FALSE	TRUE	FALSE	-	-	Only call hierarchy
18	19	2048	FALSE	FALSE	TRUE	TRUE	TRUE	Only repository structure, with file names and test directory
19	20	2048	FALSE	FALSE	TRUE	TRUE	FALSE	Only repository structure, with file names, without test directory
20	21	2048	FALSE	FALSE	TRUE	FALSE	TRUE	Only repository structure, without file names, with test directory
21	22	2048	FALSE	FALSE	TRUE	FALSE	FALSE	Only repository structure, with file names, without test directory
22	23	2048	FALSE	FALSE	FALSE	-	-	No context
23	25	4096	TRUE	TRUE	TRUE	TRUE	TRUE	All context, repository structure with file names and test directory
24	26	4096	TRUE	TRUE	TRUE	TRUE	FALSE	All context, repository structure with file names, without test directory
25	27	4096	TRUE	TRUE	TRUE	FALSE	TRUE	All context, repository structure without file names, with test directory
26	28	4096	TRUE	TRUE	TRUE	FALSE	FALSE	All context, repository structure without file names and test directory
27	29	4096	TRUE	FALSE	FALSE	-	-	Only similar code snippets
28	30	4096	FALSE	TRUE	FALSE	-	-	Only call hierarchy
29	31	4096	FALSE	FALSE	TRUE	TRUE	TRUE	Only repository structure, with file names and test directory
30	32	4096	FALSE	FALSE	TRUE	TRUE	FALSE	Only repository structure, with file names, without test directory
31	33	4096	FALSE	FALSE	TRUE	FALSE	TRUE	Only repository structure, without file names, with test directory
32	34	4096	FALSE	FALSE	TRUE	FALSE	FALSE	Only repository structure, with file names, without test directory
33	35	4096	FALSE	FALSE	FALSE	-	-	No context
34	37	8192	TRUE	TRUE	TRUE	TRUE	TRUE	All context, repository structure with file names and test directory
35	38	8192	TRUE	TRUE	TRUE	TRUE	FALSE	All context, repository structure with file names, without test directory
36	39	8192	TRUE	TRUE	TRUE	FALSE	TRUE	All context, repository structure without file names, with test directory
37	40	8192	TRUE	TRUE	TRUE	FALSE	FALSE	All context, repository structure without file names and test directory
38	41	8192	TRUE	FALSE	FALSE	-	-	Only similar code snippets
39	42	8192	FALSE	TRUE	FALSE	-	-	Only call hierarchy
40	43	8192	FALSE	FALSE	TRUE	TRUE	TRUE	Only repository structure, with file names and test directory
41	44	8192	FALSE	FALSE	TRUE	TRUE	FALSE	Only repository structure, with file names, without test directory
42	45	8192	FALSE	FALSE	TRUE	FALSE	TRUE	Only repository structure, without file names, with test directory
43	46	8192	FALSE	FALSE	TRUE	FALSE	FALSE	Only repository structure, with file names, without test directory
44	47	8192	FALSE	FALSE	FALSE	-	-	No context

TABLE B.1: Pre-experiment setups

Appendix C

Pre-Experiment Results

Experiment ID	Graph ID	Avg response time	Token amount	Avg distance
0	1	14191,2667	859,266667	0,48371005
1	2	13930,8167	859,016667	0,48046702
2	3	14388,2833	861	0,49024096
3	4	12418,4	779	0,46895971
4	5	9517,96667	504	0,47899858
5	6	7154,75	488,666667	0,47192317
6	7	6436,25	584,983333	0,48347656
7	8	6695,6	583,583333	0,47907659
8	9	6760,23333	585,65	0,48934537
9	10	6631,9	503,666667	0,48322168
10	11	4865,78333	358,666667	0,47039338
	12			
11	13	16277,6167	1564,76667	0,47067605
12	14	16997,8333	1564,76667	0,4585985
13	15	15314,95	1273,66667	0,46454866
14	16	14994,0833	1184,66667	0,46190527
15	17	11242,3667	806,666667	0,48276839
16	18	7479,35	591,666667	0,46996661
17	19	6814,03333	885,2	0,48228339
18	20	6542,75	884,783333	0,48045561
19	21	6855,38333	592,666667	0,48335468
20	22	6793,1	503,666667	0,4836561
21	23	5194,56667	358,666667	0,4697325
	24			
22	25	38684,5667	2728,15	0,46971024
23	26	37849,3667	2727,75	0,46809
24	27	18580,7667	1938,66667	0,46641519
25	28	17663,8667	1849,66667	0,47006928
26	29	13511,6667	1306,66667	0,47885297
27	30	7995,86667	756,666667	0,4670289
28	31	8577,63333	1382,71667	0,46602833
29	32	8802,88333	1382,68333	0,46881269
30	33	6504,53333	592,666667	0,47704309
31	34	6856,38333	503,666667	0,48813264
32	35	4978,08333	358,666667	0,47252239
	36			
33	37	63668,45	5249	0,40243736
34	38	52399,4167	4086,33333	0,46759784
35	39	36678,2333	2975,33333	0,46671259
36	40	30964,5167	2886,33333	0,46571078
37	41	17587,3	2331,33333	0,48029212
38	42	7586,9	768,666667	0,47138395
39	43	29797,9833	2277,66667	0,47289032
40	44	25499,9667	1703,66667	0,47171261
41	45	6712,61667	592,666667	0,47815018
42	46	6597,11667	503,666667	0,48418312
43	47	5120,23333	358,666667	0,46953017

Table C.1: Pre-experiment results Mistral

Experiment ID	Graph ID	Avg response time	Token amount	Avg distance
0	1	13104,9333	816,033333	0,51441576
1	2	12850,45	816,1	0,51400896
2	3	12514,4833	770	0,48516645
3	4	12555,3167	705	0,48554556
4	5	9041,61667	473,333333	0,4935668
5	6	5978,73333	398	0,48002784
6	7	5325,91667	501,633333	0,50158622
7	8	5211,9	500,1	0,50113983
8	9	5189,4	453,333333	0,50082825
9	10	4960,03333	388,333333	0,50018463
10	11	5035,18333	278,333333	0,49984892
	12			
11	13	20737,1	1488,61667	0,48282517
12	14	20386	1488,7	0,47706697
13	15	14261,55	1139,66667	0,47915232
14	16	14238,5833	1074,66667	0,47589053
15	17	11246,2333	740,666667	0,47634384
16	18	5845,85	500,333333	0,48658464
17	19	6059,03333	801,983333	0,49482437
18	20	6505,51667	802,083333	0,49244954
19	21	5287,18333	453,333333	0,49233271
20	22	4817,3	388,333333	0,49793197
21	23	5127,18333	278,333333	0,50341408
	24			
22	25	37220,55	2600,41667	0,49147661
23	26	38342,1833	2474,33333	0,49465089
24	27	22693,9333	1750,33333	0,47988207
25	28	17613,95	1685,33333	0,49296564
26	29	13436,9333	1251	0,48702957
27	30	6839,23333	600,666667	0,47981154
28	31	21979,25	1303,53333	0,48926466
29	32	20558,2833	1177,33333	0,49091327
30	33	5399,3	453,333333	0,49399313
31	34	4972,81667	388,333333	0,49963284
32	35	5055,8	278,333333	0,49938462
	36			
33	37	46155,85	3517,66667	0,50689312
34	38	43676,6	3130,66667	0,49970978
35	39	37126,9333	2406,66667	0,50225976
36	40	31088,0167	2341,66667	0,49523289
37	41	15816,5167	1907,33333	0,48544182
38	42	6562,81667	600,666667	0,477807
39	43	24807,2167	1564,33333	0,48857536
40	44	20879,2	1177,33333	0,48868548
41	45	5430,85	453,333333	0,49343178
42	46	5073,35	388,333333	0,49265246
43	47	5085,6	278,333333	0,49914

Table C.2: Pre-experiment results Llama3

Experiment ID	Graph ID	Avg response time	Token amount	Avg distance
0	1	13648,1	837,65	0,4990629
1	2	13390,6333	837,558333	0,49723799
2	3	13451,3833	815,5	0,48770371
3	4	12486,8583	742	0,47725263
4	5	9279,79167	488,666667	0,48628269
5	6	6566,74167	443,333333	0,4759755
6	7	5881,08333	543,308333	0,49253139
7	8	5953,75	541,841667	0,49010821
8	9	5974,81667	519,491667	0,49508681
9	10	5795,96667	446	0,49170316
10	11	4950,48333	318,5	0,48512115
	12			
11	13	18507,3583	1526,69167	0,47675061
12	14	18691,9167	1526,73333	0,46783274
13	15	14788,25	1206,66667	0,47185049
14	16	14616,3333	1129,66667	0,4688979
15	17	11244,3	773,666667	0,47955611
16	18	6662,6	546	0,47827562
17	19	6436,53333	843,591667	0,48855388
18	20	6524,13333	843,433333	0,48645257
19	21	6071,28333	523	0,4878437
20	22	5805,2	446	0,49079403
21	23	5160,875	318,5	0,48657329
	24			
22	25	37952,5583	2664,28333	0,48059343
23	26	38095,775	2601,04167	0,48137044
24	27	20637,35	1844,5	0,47314863
25	28	17638,9083	1767,5	0,48151746
26	29	13474,3	1278,83333	0,48294127
27	30	7417,55	678,666667	0,47342022
28	31	15278,4417	1343,125	0,47764649
29	32	14680,5833	1280,00833	0,47986298
30	33	5951,91667	523	0,48551811
31	34	5914,6	446	0,49388274
32	35	5016,94167	318,5	0,48595351
	36			
33	37	54912,15	4383,33333	0,45466524
34	38	48038,0083	3608,5	0,48365381
35	39	36902,5833	2691	0,48448618
36	40	31026,2667	2614	0,48047183
37	41	16701,9083	2119,33333	0,48286697
38	42	7074,85833	684,666667	0,47459548
39	43	27302,6	1921	0,48073284
40	44	23189,5833	1440,5	0,48019904
41	45	6071,73333	523	0,48579098
42	46	5835,23333	446	0,48841779
43	47	5102,91667	318,5	0,48433509

Table C.3: Pre-experiment results combined

Tokens (average), x_i	**Response time (average),** y_i	**rank**(x_i)	**rank**(y_i)	d_i	d_i^2
13648,1	837,65	24	27	-3	9
13390,6333	837,558333	23	24	-1	1
13451,3833	815,5	22	25	-3	9
12486,8583	742	20	23	-3	9
9279,79167	488,666667	10	21	-11	121
6566,74167	443,333333	5	17	-12	144
5881,08333	543,308333	16	8	8	64
5953,75	541,841667	15	11	4	16
5974,81667	519,491667	11	12	-1	1
5795,96667	446	6	5	1	1
4950,48333	318,5	1	1	0	0
18507,3583	1526,69167	33	34	-1	1
18691,9167	1526,73333	34	35	-1	1
14788,25	1206,66667	28	30	-2	4
14616,3333	1129,66667	27	28	-1	1
11244,3	773,666667	21	22	-1	1
6662,6	546	17	18	-1	1
6436,53333	843,591667	26	15	11	121
6524,13333	843,433333	25	16	9	81
6071,28333	523	12	13	-1	1
5805,2	446	6	6	0	0
5160,875	318,5	1	4	-3	9
37952,5583	2664,28333	41	41	0	0
38095,775	2601,04167	39	42	-3	9
20637,35	1844,5	36	36	0	0
17638,9083	1767,5	35	33	2	4
13474,3	1278,83333	29	26	3	9
7417,55	678,666667	18	20	-2	4
15278,4417	1343,125	31	31	0	0
14680,5833	1280,00833	30	29	1	1
5951,91667	523	12	10	2	4
5914,6	446	6	9	-3	9
5016,94167	318,5	1	2	-1	1
54912,15	4383,33333	44	44	0	0
48038,0083	3608,5	43	43	0	0
36902,5833	2691	42	40	2	4
31026,2667	2614	40	39	1	1
16701,9083	2119,33333	38	32	6	36
7074,85833	684,666667	19	19	0	0
27302,6	1921	37	38	-1	1
23189,5833	1440,5	32	37	-5	25
6071,73333	523	12	14	-2	4
5835,23333	446	6	7	-1	1
5102,91667	318,5	1	3	-2	4

TABLE C.4: Pre-experiment Spearman Correlation Tokens/Response time - Combined

Tokens (average), x_i	**Response time (average),** y_i	**rank**(x_i)	**rank**(y_i)	d_i	d_i^2
14191,2667	859,266667	23	28	-5	25
13930,8167	859,016667	22	27	-5	25
14388,2833	861	24	29	-5	25
12418,4	779	20	25	-5	25
9517,96667	504	10	23	-13	169
7154,75	488,666667	5	17	-12	144
6436,25	584,983333	12	5	7	49
6695,6	583,583333	11	10	1	1
6760,23333	585,65	13	12	1	1
6631,9	503,666667	6	9	-3	9
4865,78333	358,666667	1	1	0	0
16277,6167	1564,76667	32	32	0	0
16997,8333	1564,76667	32	33	-1	1
15314,95	1273,66667	28	31	-3	9
14994,0833	1184,66667	27	30	-3	9
11242,3667	806,666667	21	24	-3	9
7479,35	591,666667	14	18	-4	16
6814,03333	885,2	26	14	12	144
6542,75	884,783333	25	7	18	324
6855,38333	592,666667	15	15	0	0
6793,1	503,666667	6	13	-7	49
5194,56667	358,666667	1	4	-3	9
38684,5667	2728,15	40	42	-2	4
37849,3667	2727,75	39	41	-2	4
18580,7667	1938,66667	36	36	0	0
17663,8667	1849,66667	35	35	0	0
13511,6667	1306,66667	29	26	3	9
7995,86667	756,666667	18	20	-2	4
8577,63333	1382,71667	31	21	10	100
8802,88333	1382,68333	30	22	8	64
6504,53333	592,666667	15	6	9	81
6856,38333	503,666667	6	16	-10	100
4978,08333	358,666667	1	2	-1	1
63668,45	5249	44	44	0	0
52399,4167	4086,33333	43	43	0	0
36678,2333	2975,33333	42	40	2	4
30964,5167	2886,33333	41	39	2	4
17587,3	2331,33333	38	34	4	16
7586,9	768,666667	19	19	0	0
29797,9833	2277,66667	37	38	-1	1
25499,9667	1703,66667	34	37	-3	9
6712,61667	592,666667	15	11	4	16
6597,11667	503,666667	6	8	-2	4
5120,23333	358,666667	1	3	-2	4

TABLE C.5: Pre-experiment Spearman Correlation Tokens/Response time - Mistral

Tokens (average), x_i	**Response time (average)**, y_i	**rank**(x_i)	**rank**(y_i)	d_i	d_i^2
13104,9333	816,033333	25	26	-1	1
12850,45	816,1	26	25	1	1
12514,4833	770	22	23	-1	1
12555,3167	705	20	24	-4	16
9041,61667	473,333333	14	21	-7	49
5978,73333	398	9	16	-7	49
5325,91667	501,633333	17	12	5	25
5211,9	500,1	15	10	5	25
5189,4	453,333333	10	9	1	1
4960,03333	388,333333	5	2	3	9
5035,18333	278,333333	1	4	-3	9
20737,1	1488,61667	33	34	-1	1
20386	1488,7	34	32	2	4
14261,55	1139,66667	28	29	-1	1
14238,5833	1074,66667	27	28	-1	1
11246,2333	740,666667	21	22	-1	1
5845,85	500,333333	16	15	1	1
6059,03333	801,983333	23	17	6	36
6505,51667	802,083333	24	18	6	36
5287,18333	453,333333	10	11	-1	1
4817,3	388,333333	5	1	4	16
5127,18333	278,333333	1	8	-7	49
37220,55	2600,41667	42	41	1	1
38342,1833	2474,33333	41	42	-1	1
22693,9333	1750,33333	37	37	0	0
17613,95	1685,33333	36	31	5	25
13436,9333	1251	31	27	4	16
6839,23333	600,666667	18	20	-2	4
21979,25	1303,53333	32	36	-4	16
20558,2833	1177,33333	29	33	-4	16
5399,3	453,333333	10	13	-3	9
4972,81667	388,333333	5	3	2	4
5055,8	278,333333	1	5	-4	16
46155,85	3517,66667	44	44	0	0
43676,6	3130,66667	43	43	0	0
37126,9333	2406,66667	40	40	0	0
31088,0167	2341,66667	39	39	0	0
15816,5167	1907,33333	38	30	8	64
6562,81667	600,666667	18	19	-1	1
24807,2167	1564,33333	35	38	-3	9
20879,2	1177,33333	29	35	-6	36
5430,85	453,333333	10	14	-4	16
5073,35	388,333333	5	6	-1	1
5085,6	278,333333	1	7	-6	36

TABLE C.6: Pre-experiment Spearman Correlation Tokens/Response time - Llama3

Appendix D

Result Calculations

D.1 Cohen's Kappa

Tables D.1, D.2, and D.3 indicate how often raters agree on a rating (i.e. one rater rated 1, while the other rated 2 indicates both thought the response was 'bad'). As mentioned in section 4.5.2, ratings were "merged" into three distinct categories.

	Bad	**Mid**	**Good**	**Total**
Bad	14 (6.72)	4	2	20
Mid	21	17 (16.70)	23	61
Good	51	49	75 (68.36)	175
Total	86	70	100	256

TABLE D.1: Agreement table between P0 and P1

	Bad	**Mid**	**Good**	**Total**
Bad	7 (3.59)	8	19	34
Mid	5	14 (12.30)	44	63
Good	15	28	116 (111.18)	159
Total	27	50	179	256

TABLE D.2: Agreement table between P0 and P2

	Bad	**Mid**	**Good**	**Total**
Bad	22 (13.66)	21	63	106
Mid	5	9 (11.38)	42	56
Good	6	22	66 (62.79)	94
Total	33	52	171	256

TABLE D.3: Agreement table between P1 and P2

P0/P1

The total amount of agreements Σa is $14 + 17 + 75 = 106$
The expected frequencies of the number of agreements that would be expected by chance is

shown in brackets in the tables, an example calculation is: $ef = \frac{rowtotal*coltotal}{overalltotal} = \frac{86*20}{256} = 6.718.. \approx 6.62$
Getting the sum of these expected frequencies gives $\Sigma ef = 6.718... + 16.679... + 68.359... = 91.757... \approx 91.76$
With this, we can calculate the kappa: $\kappa = \frac{\Sigma a - \Sigma ef}{N - \Sigma ef} = \frac{106 - 91.757...}{256 - 91.757...} = 0.086... \approx 0.09$

P0/P2

The total amount of agreements Σa is $7 + 14 + 116 = 137$
The expected frequencies of the number of agreements that would be expected by chance is shown in brackets in the tables, an example calculation is: $ef = \frac{rowtotal*coltotal}{overalltotal} = \frac{27*34}{256} = 3.585... \approx 3.59$
Getting the sum of these expected frequencies gives $\Sigma ef = 3.585... + 12.304... + 111.175... = 127.066... \approx 127.07$
With this, we can calculate the kappa: $\kappa = \frac{\Sigma a - \Sigma ef}{N - \Sigma ef} = \frac{137 - 127.066...}{256 - 127.066...} = 0.077... \approx 0.08$

P1/P2

The total amount of agreements Σa is $22 + 9 + 66 = 97$
The expected frequencies of the number of agreements that would be expected by chance is shown in brackets in the tables, an example calculation is: $ef = \frac{rowtotal*coltotal}{overalltotal} = \frac{33*106}{256} = 13.664... \approx 13.66$
Getting the sum of these expected frequencies gives $\Sigma ef = 13.664... + 11.375 + 62.789... = 87.828... \approx 87.83$
With this, we can calculate the kappa: $\kappa = \frac{\Sigma a - \Sigma ef}{N - \Sigma ef} = \frac{97 - 87.828...}{256 - 87.828...} = 0.054... \approx 0.05$

D.2 Spearman Correlation

Table D.4 contains the average amount of tokens and the average rating for each experiment ID, together with their ranks. Using these values, the Spearman correlation can be calculated with the following equation: $\rho = 1 - \frac{6\Sigma d_i^2}{n(n^2-1)} = 1 - \frac{6*[25+9+16+1+16+4+1+4]}{8(64-1)} = 1 - \frac{456}{504} = 0.09523809523 \approx 0.10$

ID	Tokens (average), x_i	Rating (average), y_i	rank(x_i)	rank(y_i)	d_i	d_i^2
1	467,75	3,453125	1	6	-5	25
2	541,88	3,458333	2	5	-3	9
3	1646,58	3,463542	8	4	4	16
4	1646,49	3,364583	7	8	-1	1
5	1355,06	3,557292	6	2	4	16
6	1266,06	3,4375	5	7	-2	4
7	782,50	3,494792	4	3	1	1
8	652,88	3,567708	3	1	2	4

TABLE D.4: Spearman correlation values

Appendix E

Individual Agreement Tables - Cohen's Kappa

	Bad	**Mid**	**Good**	**Total**
Bad	1	0	0	1
Mid	3	2	4	9
Good	6	5	15	26
Total	10	7	19	36

TABLE E.1: Agreement table between P0 and P1 - Setup 1

	Bad	**Mid**	**Good**	**Total**
Bad	1	0	1	2
Mid	2	2	2	6
Good	3	2	11	16
Total	6	4	14	24

TABLE E.2: Agreement table between P0 and P2 - Setup 1

	Bad	**Mid**	**Good**	**Total**
Bad	4	4	8	16
Mid	0	1	4	5
Good	1	6	8	15
Total	5	11	20	36

TABLE E.3: Agreement table between P1 and P2 - Setup 1

	Bad	**Mid**	**Good**	**Total**
Bad	2	0	1	3
Mid	4	2	1	7
Good	4	8	7	19
Total	10	10	9	29

TABLE E.4: Agreement table between P0 and P1 - Setup 2

	Bad	**Mid**	**Good**	**Total**
Bad	0	1	2	3
Mid	0	2	7	9
Good	1	3	16	20
Total	1	6	25	32

TABLE E.5: Agreement table between P0 and P2 - Setup 2

	Bad	**Mid**	**Good**	**Total**
Bad	6	2	7	15
Mid	2	0	5	7
Good	2	3	8	13
Total	10	5	20	35

TABLE E.6: Agreement table between P1 and P2 - Setup 2

	Bad	Mid	Good	Total
Bad	0	1	0	1
Mid	0	1	2	3
Good	12	8	8	28
Total	12	10	10	32

TABLE E.7: Agreement table between P0 and P1 - Setup 3

	Bad	Mid	Good	Total
Bad	1	2	2	5
Mid	0	4	2	6
Good	5	5	12	22
Total	6	11	16	33

TABLE E.8: Agreement table between P0 and P2 - Setup 3

	Bad	Mid	Good	Total
Bad	2	0	11	13
Mid	1	0	6	7
Good	0	5	6	11
Total	3	5	23	31

TABLE E.9: Agreement table between P1 and P2 - Setup 3

	Bad	Mid	Good	Total
Bad	4	0	0	4
Mid	3	3	3	9
Good	6	3	6	15
Total	13	6	9	28

TABLE E.10: Agreement table between P0 and P1 - Setup 4

	Bad	Mid	Good	Total
Bad	1	3	1	5
Mid	0	1	8	9
Good	0	3	19	22
Total	1	7	28	36

TABLE E.11: Agreement table between P0 and P2 - Setup 4

	Bad	Mid	Good	Total
Bad	4	4	8	16
Mid	1	1	6	8
Good	0	0	8	8
Total	5	5	22	32

TABLE E.12: Agreement table between P1 and P2 - Setup 4

	Bad	Mid	Good	Total
Bad	2	1	0	3
Mid	3	4	5	12
Good	6	4	9	19
Total	11	9	14	34

TABLE E.13: Agreement table between P0 and P1 - Setup 5

	Bad	Mid	Good	Total
Bad	1	0	2	3
Mid	1	2	8	11
Good	0	2	19	21
Total	2	4	29	35

TABLE E.14: Agreement table between P0 and P2 - Setup 5

	Bad	Mid	Good	Total
Bad	0	5	6	11
Mid	0	0	2	2
Good	1	1	12	14
Total	1	6	20	27

TABLE E.15: Agreement table between P1 and P2 - Setup 5

	Bad	Mid	Good	Total
Bad	3	2	1	6
Mid	3	3	2	8
Good	4	8	11	23
Total	10	13	14	37

TABLE E.16: Agreement table between P0 and P1 - Setup 6

	Bad	Mid	Good	Total
Bad	0	1	3	4
Mid	1	1	7	9
Good	2	3	8	13
Total	3	5	18	26

TABLE E.17: Agreement table between P0 and P2 - Setup 6

	Bad	Mid	Good	Total
Bad	1	2	13	16
Mid	0	4	6	10
Good	0	0	7	7
Total	1	6	26	33

TABLE E.18: Agreement table between P1 and P2 - Setup 6

	Bad	Mid	Good	Total
Bad	1	0	0	1
Mid	4	0	2	6
Good	7	5	11	23
Total	12	5	13	30

TABLE E.19: Agreement table between P0 and P1 - Setup 7

	Bad	Mid	Good	Total
Bad	2	1	4	7
Mid	1	0	7	8
Good	3	5	15	23
Total	6	6	26	38

TABLE E.20: Agreement table between P0 and P2 - Setup 7

	Bad	Mid	Good	Total
Bad	3	2	4	9
Mid	1	3	4	8
Good	0	4	7	11
Total	4	9	15	28

TABLE E.21: Agreement table between P1 and P2 - Setup 7

	Bad	Mid	Good	Total
Bad	1	0	0	1
Mid	1	2	4	7
Good	6	8	8	22
Total	8	10	12	30

TABLE E.22: Agreement table between P0 and P1 - Setup 8

	Bad	Mid	Good	Total
Bad	1	0	4	5
Mid	0	2	3	5
Good	1	5	16	22
Total	2	7	23	32

TABLE E.23: Agreement table between P0 and P2 - Setup 8

	Bad	Mid	Good	Total
Bad	2	2	6	10
Mid	0	0	9	9
Good	2	3	10	15
Total	4	5	25	34

TABLE E.24: Agreement table between P1 and P2 - Setup 8

Appendix F

Major Disagreement Cases

This appendix contains all 11 major disagreements between raters (meaning one rater chose rating 1, while the other chose rating 5). Each section contains the disagreement, with it's corresponding experiment ID (see table 5.2), LLM model used, the code snippet, the question asked to the LLM, as well as the LLM response itself.

Disagreement 1 (Experiment setup 1 - Mistral)

Code Snippet

```
public String format(final JavaTypeT value) {
    if (value == null) {
        return NULL_KEYWORD;
    }
    return formatNonNull(value);
}
```

Question

What does the selected code do?

LLM Response

```
The `format` method takes a `JavaTypeT` value as input and
   returns a string. If the input value is null, it returns a
   specific keyword (`NULL_KEYWORD`). Otherwise, it calls another
    method `formatNonNull(value)` to format the non-null value.
```

Disagreement 2 (Experiment setup 2 - Mistral)

Code Snippet

```
public String format(final JavaTypeT value) {
    if (value == null) {
        return NULL_KEYWORD;
    }
```

```
    return formatNonNull(value);
}
```

Question

How can the selected code be used?

LLM Response

The `format` method is a part of the `AbstractCodec` class, which seems to be responsible for formatting values according to specific rules. This method takes a `JavaTypeT` value as input and returns a formatted string.

Based on the context, it appears that this method is used to format values in a Cassandra JDBC wrapper. The method checks if the input value is null, and if so, returns a predefined constant `NULL_KEYWORD`. If the value is not null, it calls another method `formatNonNull` to perform the actual formatting.

The callers of this method seem to be various test classes that are testing different codec implementations. They pass in either null or non-null values to the `format` method and verify the expected output.

In summary, the selected code can be used to format values according to specific rules in a Cassandra JDBC wrapper. It's primarily used for testing purposes, but it could also be used as a building block for more complex formatting logic in other parts of the application.

Disagreement 3 (Experiment setup 3 - Mistral)

Code Snippet

```
public <T> T unwrap(final Class<T> iface) throws SQLException {
    if (isWrapperFor(iface)) {
        return iface.cast(this);
    } else {
        throw new SQLException(String.format(NO_INTERFACE, iface.
            getSimpleName()));
    }
}
```

Question

What is being assigned to the variables in the selected code?

LLM Response

In the selected code snippet, `unwrap` method is assigning an object of type `T` to a variable. The assignment is happening inside an `if` statement, which checks if the current object (`this`) is an instance of the interface `iface`. If it is, then the method returns `iface.cast(this)`, effectively casting the current object to the specified interface type `T`.

Disagreement 4 (Experiment setup 4 - Mistral)

Code Snippet

```
boolean acquire() {
    while (true) {
        final int ref = this.references.get();
        if (ref < 0) {
            // We raced with the release of the last reference,
               the caller will need to create a new session.
            LOG.debug(\"Failed to acquire reference to {}.\",
               this.cacheKey.get(URL_KEY));
            return false;
        }
        if (this.references.compareAndSet(ref, ref + 1)) {
            LOG.debug(\"Acquired reference to {}, new count =
               {}.\", this.cacheKey.get(URL_KEY), ref + 1);
            return true;
        }
    }
}
```

Question

How can the selected code be used?

LLM Response

The `acquire()` method is used to acquire a reference to a Cassandra session. It's part of the `SessionHolder` class, which is responsible for managing sessions in the Cassandra JDBC wrapper.

This method can be used when you want to ensure that only one thread or process has access to a particular Cassandra session at any given time. The method uses a while loop and a compareAndSet operation on a reference counter to achieve this.

```
In other words, `acquire()` is a synchronization mechanism that
    allows multiple threads to share the same Cassandra session,
    but ensures that only one thread can use it at a time.
```

Disagreement 5 (Experiment setup 6 - Mistral)

Code Snippet

```
public <T> T unwrap(final Class<T> iface) throws SQLException {
    if (isWrapperFor(iface)) {
        return iface.cast(this);
    } else {
        throw new SQLException(String.format(NO_INTERFACE, iface.
            getSimpleName()));
    }
}
```

Question

How can the selected code be used?

LLM Response

```
This code snippet is part of an AbstractConnection class in a
    Cassandra JDBC wrapper. It provides an implementation for the
    unwrap method, which is part of the javax.sql WrapperFactory
    interface.

The unwrap method takes a Class parameter representing the
    interface type to be unwrapped. If this connection instance
    implements or wraps the specified interface, it returns itself
     cast to the specified interface type. Otherwise, it throws a
    SQLException with a message indicating that no interface was
    found.

In your onboarding process, you can use this code as part of
    implementing a Cassandra JDBC wrapper for your application.
    The unwrap method is useful when working with different
    database interfaces or APIs, allowing you to dynamically
    switch between them based on the required functionality.
```

Disagreement 6 (Experiment setup 7 - Mistral)

Code Snippet

```
public String format(final JavaTypeT value) {
    if (value == null) {
        return NULL_KEYWORD;
```

```
    }
    return formatNonNull(value);
}
```

Question

What is being assigned to the variables in the selected code?

LLM Response

```
The variable `value` is an instance of `JavaTypeT`, which is a
    generic type. In the first line, we check if `value` is null
    or not. If it is null, then the string constant `NULL_KEYWORD`
     is assigned to the return value. Otherwise, the method `
    formatNonNull(value)` is called and its return value is
    assigned to the return variable.
```

Disagreement 7 (Experiment setup 2 - Llama3)

Code Snippet

```
if (isCqlType(columnIndex, DataTypeEnum.LIST)) {
    final ListType listType = (ListType) cqlDataType;
    final DataType elementsType = listType.getElementType();
    final List<?> resultList;

    if (elementsType instanceof TupleType) {
        resultList = this.currentRow.getList(columnIndex - 1,
        TypesMap.getTypeForComparator(DataTypeEnum.TUPLE.
            asLowercaseCql()).getType());
    } else {
        resultList = this.currentRow.getList(columnIndex - 1,
        TypesMap.getTypeForComparator(elementsType.asCql(false,
            false)).getType());
    }
    if (resultList == null) {
        return null;
    }
    return new ArrayList<>(resultList);
}
```

Question

What is being assigned to the variables in the selected code?

LLM Response

```
In the given code snippet, the following variables are being
    assigned:

1. `columnIndex`: An integer representing the index of a column
    in a Cassandra result set.
2. `DataTypeEnum.LIST`: A constant representing the data type of
    a list in Cassandra.
3. `cqlDataType`: An object of type `DataType` that represents
    the Cassandra data type for the given column.
4. `listType`: An object of type `ListType` that holds
    information about the list, such as its element type.
5. `elementsType`: The data type of the elements in the list.
6. `resultList`: A Java List that will hold the values of the
    list column for the current row.
7. `currentRow`: An object representing a single row in the
    Cassandra result set.
8. `TypesMap.getTypeForComparator(DataTypeEnum.TUPLE.
    asLowercaseCql())` or `TypesMap.getTypeForComparator(
    elementsType.asCql(false, false))`: The Java type that
    corresponds to the given Cassandra data type for comparison
    purposes.
```

Disagreement 8 (Experiment setup 3 - Llama)

Code Snippet

```
for (final CompletionStage<AsyncResultSet> future : futures) {
    final AsyncResultSet asyncResultSet = CompletableFutures.
       getUninterruptibly(future);
    final com.datastax.oss.driver.api.core.cql.ResultSet rows;
    if (asyncResultSet.hasMorePages()) {
        rows = new MultiPageResultSet(asyncResultSet);
    } else {
        rows = new SinglePageResultSet(asyncResultSet);
    }
    results.add(rows);
}
```

Question

What does the selected code do?

LLM Response

```
The selected code is a part of a Java class named `
    CassandraStatement`. It is located in the file `{PATH\_TO}/
    CassandraStatement.java`.
```

This code snippet is responsible for processing the result set of a SQL query or update statement. It is called from various methods such as `executeQuery(String)`, `executeUpdate(String)`, and `execute(String)` in the `CassandraStatement` class. The ultimate caller of these methods can be traced back to different test cases in various unit tests files.

The code starts by checking if the result set is empty or not. If it's empty, an exception is thrown. Otherwise, it iterates through each row in the result set and processes it based on its data type. For instance, if the data type is a `Date`, it converts it to a `java.util.Date` object. Similarly, for other data types like `Time`, `Timestamp`, or custom user-defined types (UDTs), appropriate conversions are performed.

The processed data is then returned as an array of objects, which can be further used by the calling method or test case to assert expected results.

Disagreement 9 (Experiment setup 3 - Llama)

Code Snippet

```
boolean acquire() {
    while (true) {
        final int ref = this.references.get();
        if (ref < 0) {
            // We raced with the release of the last reference,
               the caller will need to create a new session.
            LOG.debug(\"Failed to acquire reference to {}.\",
               this.cacheKey.get(URL_KEY));
            return false;
        }
        if (this.references.compareAndSet(ref, ref + 1)) {
            LOG.debug(\"Acquired reference to {}, new count =
               {}.\", this.cacheKey.get(URL_KEY), ref + 1);
            return true;
        }
    }
}
```

Question

How can the selected code be used?

LLM Response

The selected code is a part of a Java class named `SessionHolder` located in the file `{PATH_TO}/SessionHolder.java`. This code defines a method named `acquire()` which is used to acquire a reference to an instance of `SessionHolder`.

The `acquire()` method uses a `while (true)` loop to repeatedly attempt to acquire a reference by checking the current value of `references` and then attempting to increment it using `compareAndSet()`. If the acquisition is successful, the method logs a message indicating that the reference was acquired and returns `true`. If the acquisition fails, the method logs an error message and returns `false`.

This code is likely used in a multi-threaded environment where multiple threads may attempt to acquire the same instance of `SessionHolder`. By using the `compareAndSet()` method, the first thread to successfully increment the reference count will be able to acquire the instance, while other threads will need to retry.

The similar code snippets you provided from files like `MetadataResultSetsUnitTest.java`, `ManagedPreparedStatement.java`, and `ByteBufferUtil.java` are not directly related to the selected code, but they do provide context about the Java project and its structure. The repository structure you provided also gives additional context about the project organization.

The call hierarchy shows that the `acquire()` method is called by the `connect(String, Properties)` method in the `CassandraDriver.java` file. This means that when a connection is established using the `CassandraDriver`, an instance of `SessionHolder` is acquired for use during the session.

Disagreement 10 (Experiment setup 6 - Llama)

Code Snippet

```
try {
    while ((nRead = inputStream.read(data, 0, data.length)) !=
       -1) {
        buffer.write(data, 0, nRead);
    }
} catch (final IOException e) {
    throw new SQLNonTransientException(e);
}
```

Question

What is being assigned to the variables in the selected code?

LLM Response

In the given code snippet, the variable `nRead` is assigned the value returned by the `read()` method call on an InputStream object named `inputStream`. The byte array `data` is being filled with the data read from the input stream in each iteration of the while loop. The variable `buffer` is a ByteBuffer that is being written to with the data read from the input stream.

The exception thrown in the catch block is an IOException, which is then wrapped and re-thrown as an SQLNonTransientException. This exception is likely to be handled by higher-level code, possibly in a try-catch block at a call site further up the call stack.

Regarding the context you've provided: The selected code snippet is found in the file `CassandraPreparedStatement.java`, which is part of a Java project with several other files, including `CassandraResultSet.java` and `ConversionsUtil.java`. These files contain similar try-catch blocks for handling InputStreams and exceptions. The repository structure suggests that this code is part of a larger Java project related to Cassandra database interaction.

The call hierarchy shows that the method `setBlob()` is the entry point for accessing the selected code snippet, with multiple callers including `JdbcRegressionUnitTest.java`, `ManagedPreparedStatement.java`, and `CassandraPreparedStatement.java`.

Disagreement 11 (Experiment setup 6 - Llama)

Code Snippet

```
if (isCqlType(columnIndex, DataTypeEnum.LIST)) {
    final ListType listType = (ListType) cqlDataType;
    final DataType elementsType = listType.getElementType();
    final List<?> resultList;

    if (elementsType instanceof TupleType) {
        resultList = this.currentRow.getList(columnIndex - 1,
        TypesMap.getTypeForComparator(DataTypeEnum.TUPLE.
            asLowercaseCql()).getType());
    } else {
```

```
            resultList = this.currentRow.getList(columnIndex - 1,
            TypesMap.getTypeForComparator(elementsType.asCql(false,
                false)).getType());
        }
        if (resultList == null) {
            return null;
        }
        return new ArrayList<>(resultList);
    }
```

Question

What is being assigned to the variables in the selected code?

LLM Response

In the given code snippet, the following variables are being assigned:

1. `columnIndex`: An integer representing an index of a column.
2. `DataTypeEnum.LIST`: A constant value representing a list data type.
3. `cqlDataType`: A DataType object that represents the Cassandra CQL data type for the given column.
4. `listType`: A ListType object that holds information about the list, such as its element type.
5. `elementsType`: The DataType object representing the type of elements in the list.
6. `resultList`: A List interface that holds a collection of values for the given column.
7. `currentRow`: An object that represents a single row in a result set.
8. `TypesMap`: A map-like object used to convert between different data types and their CQL representations.
9. `getTypeForComparator`: A method from the TypesMap interface that returns a DataType object based on its CQL representation.

Appendix G

Classification of badly rated responses

P0		**P1**	
Classification	Explanation	Classification	Explanation
incorrect	not exclusive access	incorrect	It's not about exclusive access
incomplete	should also include contactPoints as a variable.	Not helpful	Technically it's an answer, but it's the worst. It misses context about what contactPoints is
hallucination	"passing in a 'SQLException'" is weird	incorrect	Incorrect description on usage (like prev remark says)
incorrect	NULL_KEYWORD is not assigned and it is a constant	incorrect	It's not a variable and it's not assigned with a value. The answer 'is the other way around'.
incorrect	it takes one node from the stream of nodes (the first) to get the data, not each node	incorrect	It only prints it from the first node (findFirst())
incorrect	statements is not being assigned to statement <- incorrect	incorrect	incorrect second assignment description
incorrect	'are being assigned values' is incorrect (they were already assigned)	incorrect	second paragraph is right, first is incorrect
hallucination	NULL_KEYWORD is not a keyword (hallucination because of KEYWORD ref)	incorrect	agree with the other remark, but I still find it to be useful
helpful, but not answering the question	helpful, but not answering the question ("how can the code be used")	Not useful	Like my earlier comment. The how is way too vague/open and that's the question.
incorrect	NULL_KEYWORD being assigned is wrong	incorrect	value will not be assigned
incorrect	very wordy, the first branch doesn't assign	incorrect	both; no assignment and also wordy/lengthy response
prompt-leaking	it is not primarily used for testing purposes and also not for 'complex' formatting logic (it is not very complex)	Uncertain	I initially said not for testing purposes, but t might be after all. Not sure :)
incorrect	is assigning an object of type 'T' to a variable is wrong	incorrect	No assignment, but explanation is still useful
incorrect	'this is being assigned to the variable' <- incorrect	incorrect	Weird explanation about assignment
hallucination	this method is called when a new developer want to use the codebase LOL	hallucination	'called when a new developer wants to use the codebase' ->quote that in the thesis :D
prompt-leaking	1. onboarding new developers. prompt leaking	prompt-leaking	onboarding new devs, again very funny leak

P0		**P1**	
Classification	Explanation	Classification	Explanation
incomplete	should have mentioned variable contactPoints and props variable.	incorrect	It's only about the happy flow, not about what if the assumptions are violated
hallucination	the last paragraph is just blabbing with no real content. the first paragraph is wordy	Not useful	I don't like the hypothetical conjecture (not without refs), it's very wordy and not very precise.
hallucination	not really about variables being assigned. ContactPoints is a fabrication	False Positive	seems to be OK, maybe incorrectly judged?
prompt-leaking	prompt leaking, value is not being assigned	incorrect	corp key leak, 'value is assigned'.
hallucination	second paragraph is garbage	incorrect	not an assignment to a var
incorrect	for each node is wrong. it only uses one node (the first).	hallucination	Not for each node, i don't like the 'does it make sense?' trailing question and the hypotheticals at the end.
incorrect	not about one thread getting exclusive access.	incorrect	Not about exclusive access
incorrect	not about variables being assigned	incorrect	not answering the prompt, weird sentence ('being removed from').
hallucination	first paragraph tries to explain the method call as a 'variable' in a very wordy way. the second paragraph is not relevant	incorrect	Not about assignments. Decent explanation though.
prompt-leaking	'in your onboarding process' <- prompt leaking. the first paragraph could be dropped	Not helpful	Very vague, imprecise and wordy. Doesn't help at all.
good	This is actually pretty good	False positive	On second thought, pretty good.
incorrect	'ensure only one thread' <- incorrect	incorrect	Not about exclusive access
incorrect	method calls that 'are being assigned' is wrong	incorrect	It's not an assignment, but close enough.
incorrect	not about thread safety	incorrect	Not related to thread safety indeed
hallucination	"assigned back to the return value" <- weird	incorrect	Class<T>refers not to a class object representing some interface type, but to classes or interfaces _themselves_ in the JVM.
incorrect	no new instance	incorrect	No new instance indeed
incorrect	'only one instance of a session' <- incorrect	incorrect	Exclusivity of first paragraph is wrong, it's not about exclusive access

P0		**P1**	
Classification	Explanation	Classification	Explanation
hallucination	first paragraph is wrong (isDbaasConnection is not being assigned). "assigned to a new exception" <- weird	incorrect	Wall of text, a lot of noise unrelated to question
incorrect	3 and 4 are not variable being assigned	incorrect	plenty of refs to non-assignments
hallucination	return variable <- weird.	incorrect	return value assigned to the return variable
hallucination	setX(i. value) ???	hallucination	setX is hallucinated
incorrect	the last paragraph is wrong	incorrect	ref <0 explanation incorrect, last paragraph wrong
incorrect	point 1, 2, 3 are not variables being assigned	incorrect	Agree with existing statement
incorrect	"is an instance of DriverConfigLoader.programmaticBuilder" is incorrect	incorrect	agree with current assessments
hallucination	a class 'iface' <- incorrect. the last paragraph is babbling	incorrect	Wrong, assignment to variable, T any iface extending Connection
hallucination	the order of sentences makes it hard to understand. there is some duplication.	Complicated response	The structure of the paragraph is indeed a bit complicated, but apart from that OK. I actually don't mind this one that much.
hallucination	very generic about variables being set. not answering the question. "each variable definition" <- weird	incorrect	Confusing and not answering the question, lengthy
incorrect	last paragraph is wrong	incorrect	Only one thread is incorrect. Last paragraph still looks wrong ('internal implementation detail').
prompt-leaking	last paragraph is leaking the prompt and non-relevant variables	prompt-leaking	Quite a bit of prompt leaking
prompt-leaking	prompt leaking. ultimate caller	hallucination	Leaks corp key, I don't like all the test code refs. It also doesn't start with checking whether the set is empty. I guess incorrect + hallucination + leakage
incorrect	rows is directly assigned so second paragraph is wrong	incorrect	Wrong, see existing comments i agree
prompt-leaking	prompt leaking, tests are not relevant	prompt-leaking	Too wordy, leakage of corp key, tests (assuming no hallucination).

P0		**P1**	
Classification	Explanation	Classification	Explanation
prompt-leaking	last paragraph 'in the test cases' is weird	prompt-leaking	I don't like all the testcases added to the prompt. I guess it could be called leakage as it is not related to the promp.t
prompt-leaking	5. is wrong	False positive	Corp key leaking, 5. is wrong (leak or hallucination?). I also don't like the genericity of the last paragraph
prompt-leaking	'onboard a new developer' is leaking. the first paragraph is a bit wordy and therefor harder to understand	prompt-leaking	It's a bit wordy, but indeed the second paragraph is very weird. It's either hallucination or prompt leakage
hallucination	it a mixed bag and not focused on variable assignment	Not helpful	Not answering the question
prompt-leaking	leaks prompt ub85uk	incorrect	leaks corp key, verbose, leaks prompt
prompt-leaking	it leaks a lot of variables that are not in the snippet	prompt-leaking	references prompt leak context (or maybe hallucinates; stmt3 etc)
prompt-leaking	9 and 10	prompt-leaking	9 and ten are not in scope
prompt-leaking	ub85uk prompt leaking. paging not mentioned	prompt-leaking	corp key leak, don't like the unit test refs, incorrectly mentioning checking if empty and then throwing exception
incomplete + prompt leaking	leaks ub85uk. paging missing	incorrect	leaks corp key. Decent explanation but indeed not about pagination present in code
incorrect	SQLRecoverableException is a type, not an instance. point 1 is weird and confusing	incorrect	no assigments, but useful nonetheless
incorrect	driverConfigLoaderBuilder cannot be an 'instance' of a method call. the actual type is ProgrammaticDriverConfigLoaderBuilder	incorrect	Agree with existing feedback
prompt-leaking	prompt leaking ub85uk. 'the context provided'. last paragraph is bla bla	incorrect	corp key leak, only a vague ref about usage at the end, too long and imprecise
hallucination	"is assigned to the string variable" <- WHAT?	incorrect	'...is assigned to the string var' ->that's not right.

P0		**P1**	
Classification	Explanation	Classification	Explanation
hallucination	point 1 till 4 are on the wrong level of 'usage'	incorrect	VersionedMetaData is an interface, not an object with an accompanying constructor.
prompt leaking, incorrect,hallucination	contactList, schemaName,catalogName hallucinated (in the prompt as context?). property user does not exists	hallucination	It references non existing variables
prompt-leaking	variables are mentioned that are not in the snippet	hallucination	looks like it hallucinates the max/min etc props
incorrect	not waiting because it is async	incorrect	"as a single transaction" –>No, they are separate queries that do not share transactional boundaries
incorrect	event.getSQLException() is not a variable	incorrect	the sentence about event.getSQLException() is weird / incorrect on multiple levels.
prompt-leaking	leaking prompt in the first, third and fourth paragraph.	prompt-leaking	prompt leak, too much context
prompt-leaking	a lot of context leaking in the prompt just to explain what the code does	prompt-leaking	prompt leakage
prompt-leaking	the section 'variable 'statement' in the test cases is really weird' probably because of the context?	prompt-leaking	Presumed prompt leaking due to the test references.
incorrect	8 and 9 are wrong	incorrect	At least 9 indeed wrong (ArrayList<Object>)
prompt-leaking	leaks ub85uk. last paragraph leaks prompt. doesn't really say something about usage	prompt-leaking	I don't like the last paragraph and the rest is very verbose and high level. Overall not helpful but also not terrible
incorrect	1. is wrong (not a variable)	incorrect	1. is not an assignment
incorrect	no variables are assigned	incorrect	result is assigned to the return value – technically it's just returned.
prompt-leaking	a lot of prompt is leaking through	prompt-leaking	Corp key leaked, I also don't care about the files provided to the context. Last paragraph seems to be what should have been the second paragraph, dropping everything in between
prompt-leaking	ub85uk leaking. 'the context provided'	prompt-leaking	corp key leak, first paragraph about exception incorrect,

P0		**P1**	
Classification	Explanation	Classification	Explanation
prompt-leaking	too much prompt leaking (last two paragraphs). difficult structure	prompt-leaking	hard to follow, difficult structure, context leaking fully agree
hallucination	why mention 3? 2. "is assigned to this methods return value" <- weird	incorrect	It incorrectly interprets the debug log statement and doesn't understand CAS
incorrect	TypesMap?. point 9?	incorrect	not all assignments
incorrect	'is being discussed' <- weird. isClosed is not a variable.	incorrect	It's very wordy and imprecise. For example, 3. 'the var isClosed is bool and set to true once all statements have been closed' is bordering hallucination. Also some of them are not answers to the question
incorrect	last paragraph is wrong - it is not an independent utility function	hallucination	Weird response
prompt-leaking	prompt leaking? byteArray and x are not in snippet	incorrect	byteArray and x are not in scope
helpful, but not answering the question	helpful, but not answering the question	incorrect	Agree, not about variables
incorrect	count <0 doesn't mean another thread has acquired the reference: wrong	incorrect	Wrong, agree with prior remark.
incorrect	paragraph 2 "DriverConfigLoader.programmaticBuilder()" is wrong	incorrect	Agree
prompt-leaking	ub85uk leaking. too much context about test classes. too much discussion about SessionHolder	prompt-leaking	Same as existing; corp key leakage, refs to tests classes and sessionholder ref too detailed and noisy for the question
incorrect	the return value is not really an assignment	incorrect	assigned to return value incorrect
hallucination	a lot of irrelevant details in first paragraph, point 4 for example	Not helpful	too verbose, too much detail and at the same time missing the serializer.
prompt-leaking	prompt leak - paragraph 2 is not very helpful (hard to follow)	prompt-leaking	corp key. I don't like the test refs (leakage?). Explanation is a bit fuzzy.

P0		**P1**	
Classification	Explanation	Classification	Explanation
hallucination	"iterates through each variable definition" <- weird. also last paragraph "unset variable definitions"	incorrect	Explanation is decent, but until halfway point of the snippet. Also doesn't answer the question
prompt-leaking	'Regarding your question about the variables in the context you provided' <- prompt leaking. The last paragraph 'the variables in your question' is weird	incorrect	The part about SQLNonTransientException is incorrect. The later paragraphs about test data is not asked for and not helpful either.

TABLE G.1: Initial major disagreements

Appendix H

Pre-Tasks Survey

User Study Pre-Task Survey

Short pre-task interview just to get some basic information

* Required

* This form will record your name, please fill your name.

1. How many professional years of experience do you have as a developer? *

2. How often do you onboard in a new codebase? *

   This could be new codebases within your teams in <<company>>, but can also include (open source) codebases you contribute to outside of work. Please formulate your answer like "Once every <x>"

3. What methods do you usually use to understand a new codebase? *

   - [ ] Ask an expert developer to explain the code to me
   - [ ] Read documentation
   - [ ] Run the code yourself
   - [ ] Read the tests for a piece of code
   - [ ] Use google
   - [ ] Use an LLM tool
   - [ ] Read the code itself
   - [ ] Other

4. How much experience do you have with using LLMs for code-related tasks? *

I.e. Github Copilot, ChatGPT

- ◯ I don't know what an LLM is
- ◯ I have heard of LLMs, but have never used them
- ◯ I rarely use LLMs for code related tasks
- ◯ I occasionally use LLMs for code related tasks
- ◯ I use LLMs for code related tasks often

5. Please indicate how often you use LLM tools for the following tasks *

	Never	Rarely	Sometimes	Often	Very often
Generating new code	◯	◯	◯	◯	◯
Understanding code	◯	◯	◯	◯	◯
Generating documentation	◯	◯	◯	◯	◯
Fixing bugs	◯	◯	◯	◯	◯
Summarising code	◯	◯	◯	◯	◯

6. How familiar are you with the VSCode IDE? *

- ◯ I have never heard of VSCode
- ◯ I have heard of VSCode, but never used it
- ◯ I have tried VSCode in the past, but don't use it anymore
- ◯ I use VSCode, but haven't been using it very long
- ◯ I am a long-time user of VSCode

7. How experienced are you with programming in Java? *

- ◯ I have never heard of Java
- ◯ I have heard of Java, but have never used it
- ◯ I have used Java a little bit
- ◯ I am comfortable implementing most tasks in Java
- ◯ I am a Java expert

Appendix I

Inter-Task & Post-Task Questions

User Study Task / Post-Task Survey

* Required

* This form will record your name, please fill your name.

Introduction

Hi!

Thank you for investing your time in this user study. This survey consists of questions about the individual tasks you will be completing, these questions will have to be answered after completing each task (15 minutes each). The last part contains a few questions about the LLM tool in general.

Cheers,
Koen

Please complete task 1 before continuing to the next questions

Task 1

1. For task 1, answer the following statements: *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The task was difficult to understand.	◯	◯	◯	◯	◯
The necessary changes were difficult to implement.	◯	◯	◯	◯	◯
The task was mentally challenging.	◯	◯	◯	◯	◯
I had enough time to complete the task.	◯	◯	◯	◯	◯
I was able to complete the task.	◯	◯	◯	◯	◯
I am confident that I understand the code related to this task.	◯	◯	◯	◯	◯
I am confident in the answer I produced.	◯	◯	◯	◯	◯

2. If you haven't completed the task, please write down some small steps how you would complete it if you had more time.

3. Please explain the essence of the code you looked at in two sentences. *

4. If you have any additional remarks regarding task 1, please leave them here.

Please complete task 2 before continuing to the next questions

Task 2

5. For task 2, answer the following statements: *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The task was difficult to understand.	◯	◯	◯	◯	◯
The necessary changes were difficult to implement.	◯	◯	◯	◯	◯
The task was mentally challenging.	◯	◯	◯	◯	◯
I had enough time to complete the task.	◯	◯	◯	◯	◯
I was able to complete the task.	◯	◯	◯	◯	◯
I am confident that I understand the code related to this task.	◯	◯	◯	◯	◯
I am confident in the answer I produced.	◯	◯	◯	◯	◯

6. If you haven't completed the task, please write down some small steps how you would complete it if you had more time.

7. Please explain the essence of the code you looked at in two sentences. *

8. If you have any additional remarks regarding task 2, please leave them here.

Please complete task 3 before continuing to the next questions

Task 3

9. For task 3, answer the following statements: *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The task was difficult to understand.	◯	◯	◯	◯	◯
The necessary changes were difficult to implement.	◯	◯	◯	◯	◯
The task was mentally challenging.	◯	◯	◯	◯	◯
I had enough time to complete the task.	◯	◯	◯	◯	◯
I was able to complete the task.	◯	◯	◯	◯	◯
I am confident that I understand the code related to this task.	◯	◯	◯	◯	◯
I am confident in the answer I produced.	◯	◯	◯	◯	◯

10. If you haven't completed the task, please write down some small steps how you would complete it if you had more time.

11. Please explain the essence of the code you looked at in two sentences. *

12. If you have any additional remarks regarding task 3, please leave them here.

Please complete task 4 before continuing to the next questions

Task 4

13. For task 4, answer the following statements: *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The task was difficult to understand.	◯	◯	◯	◯	◯
The necessary changes were difficult to implement.	◯	◯	◯	◯	◯
The task was mentally challenging.	◯	◯	◯	◯	◯
I had enough time to complete the task.	◯	◯	◯	◯	◯
I was able to complete the task.	◯	◯	◯	◯	◯
I am confident that I understand the code related to this task.	◯	◯	◯	◯	◯
I am confident in the answer I produced.	◯	◯	◯	◯	◯

14. If you haven't completed the task, please write down some small steps how you would complete it if you had more time.

15. Please explain the essence of the code you looked at in two sentences. *

16. If you have any additional remarks regarding task 4, please leave them here.

General questions

17. Order tasks from least to most difficult *

- Task 1
- Task 2
- Task 3
- Task 4

18. For the LLM-based tasks, please rate how much you agree with the following statements: *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Learning to use the LLM tool was easy for me.	◯	◯	◯	◯	◯
The LLM tool helped me more quickly understand the code.	◯	◯	◯	◯	◯
The LLM tool gave helpful answers.	◯	◯	◯	◯	◯
The LLM tool gave confusing results.	◯	◯	◯	◯	◯
The LLM tool took too long to respond.	◯	◯	◯	◯	◯
I had trouble completing the tasks in the allotted time.	◯	◯	◯	◯	◯
I had trouble understanding the tasks at hand.	◯	◯	◯	◯	◯
Without the LLM tool, the tasks would have taken longer to complete.	◯	◯	◯	◯	◯
The LLM tool was easy to interact with.	◯	◯	◯	◯	◯
It was easy for me to get the answers I needed from the LLM tool.	◯	◯	◯	◯	◯

19. For the non-LLM based tasks, please rate how much you agree with the following statements: *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Using traditional comprehension techniques was easy for me	◯	◯	◯	◯	◯
Using traditional methods helped me more quickly understand the code	◯	◯	◯	◯	◯
Using traditional methods gave me helpful answers	◯	◯	◯	◯	◯
Using traditional methods gave me confusing results	◯	◯	◯	◯	◯
Using traditional methods took too long to find results	◯	◯	◯	◯	◯
I had trouble completing the tasks in the allotted time	◯	◯	◯	◯	◯
I had trouble understanding the tasks at hand.	◯	◯	◯	◯	◯
Without traditional methods, the tasks would have taken longer to complete	◯	◯	◯	◯	◯
Traditional methods were easy to interact with	◯	◯	◯	◯	◯
It was easy for me to get the answers I needed from traditional methods	◯	◯	◯	◯	◯

20. How valuable would the LLM tool be for your work if it could be added to your development environment? *

	Not valuable	Limited value	Average value	Valuable	Very valuable
In its current state	◯	◯	◯	◯	◯
With a larger scope (i.e. faster/larger model)	◯	◯	◯	◯	◯

21. Please shortly explain why *

22. If you have any additional remarks, please leave them here