Purpose Enhanced Reasoning through Iterative Prompting: Uncover Latent Robustness of ChatGPT on Code Comprehension

Yi Wang, Qidong Zhao, Dongkuan Xu and Xu Liu
North Carolina State University
ywang336@ncsu.edu, qzhao24@ncsu.edu, dxu27@ncsu.edu, xliu88@ncsu.edu

Abstract

Code comments are crucial for gaining in-depth insights to facilitate code comprehension. The key to obtaining these insights lies in precisely summarizing the main purpose of the code. Recent approaches on code comment generation lie in prompting large language models (LLMs) such as ChatGPT, instead of training/fine-tuning specific models. Although ChatGPT demonstrates an impressive performance in code comprehension, it still suffers from robustness challenges in consistently producing high-quality code comments. This is because ChatGPT prioritizes the semantics of code tokens, which makes it vulnerable to commonly encountered benign perturbations such as variable name replacements. This study proposes a modular prompting paradigm Perthept to effectively mitigate the negative effects caused by such minor perturbations. Perthept iteratively enhances the reasoning depth to reach the main purpose of the code. Perthept demonstrates robustness under the scenario where there is stochasticity or unreliability in ChatGPT’s responses. We give a comprehensive evaluation across four public datasets to show the consistent robustness improvement with our proposed methodology over other models.

1 Introduction

Code comment generation has attracted extensive attention in recent years. High-quality code summaries provide deep insights into code behavior, facilitating code comprehension. Automatic code comment generation has been widely used in program comprehension [Sridhara et al.2010], software development [de Souza et al.2005], bug fixing [Panicella et al.2016], and software maintenance [He2019].

Large language models (LLMs) with exceedingly large training datasets and optimized training approaches have shown remarkable abilities in various generation tasks including code comment generation, showcasing extensive generalization and advanced semantics comprehension ability. Programmers can use a straightforward prompt like “summarize the code” to interact with the LLM and receive a satisfactory code summary. Therefore, there is a growing trend of prompting LLMs (e.g., ChatGPT1, GPT-42) for generating code summaries. However, LLMs tend to generate code summaries based on the semantic information of code tokens, which results in missing deep insights into code behavior and suffering robustness issues in consistently producing high-quality code summaries when a minor perturbation (e.g., function/variable/argument name replacement) is introduced into the source code. Take ChatGPT as an example, as shown in Figure 1, when the semantic-rich argument name packed is transformed into a commonly used name p, as shown on the right side, ChatGPT fails to identify that it is an unpacking algorithm. This case indicates the robustness issue of ChatGPT in code comprehension.

Figure 1: This figure provides a case of the decrease in the quality of code comment generation. The original codes on the left side represent an unpacking algorithm, and the code summary generated by ChatGPT includes the keyword “unpack”. When argument name packed is transformed into a commonly used name p, as shown on the right side, ChatGPT fails to identify that it is an unpacking algorithm. This case indicates the robustness issue of ChatGPT in code comprehension.

1https://openai.com/blog/chatgpt
2https://openai.com/research/gpt-4
ChatGPT formattedly generates lengthy and useless descriptions of the code.

Since fine-tuning ChatGPT is costly, prompt design serves as a practical solution to effectively address this robustness issue. We propose Perthept (Purpose enhanced reasoning through iterative prompting), a prompting framework to uncover the latent capability of large language models to generate robust and high-quality code summaries. Perthept is composed of two prompting paradigms, Chain-of-Structure and Reasoning-Enhancement. We develop Chain-of-Structure prompting with a special-designed granularity of information detail, that prompts ChatGPT to shift its attention from unreliable semantics information to reliable structure information, facilitating purpose reasoning. Meanwhile, Chain-of-Structure helps to ensure no information loss. In addition, we design a novel paradigm Reasoning-Enhancement to prompt ChatGPT to perform a self-evaluation of the extracted main purpose of the code. Therefore, ChatGPT can further refine its predictions at a higher reasoning level. The overall workflow is shown in Figure 2. This workflow not only provides deep insights for code summaries, but also mitigates the negative influence caused by the inherent unreliability of ChatGPT. Finally, we evaluate Perthept on the TL-CodeSum datasets [Hu et al. 2018b], the CodeSearchNet Java datasets [Husain et al. 2019], the CodeSearchNet Python datasets [Husain et al. 2019], and the Python datasets [Wan et al. 2018] respectively. In order to exclude samples that do not align with the focus of this study, which is reasoning the high-level main purpose of the code, we develop a filter algorithm to extract the potential reasoning-required code samples from these well-studied datasets because most code samples are simple and do not require high-level reasoning. We summarize the contributions as follows:

- We identify and assess the purpose-reasoning ability of ChatGPT for code comprehension. Extensive experiments show that purpose-reasoning can be effectively utilized to uncover ChatGPT’s inherent capacity in producing robust and insightful code summaries.
  - We introduce a modular framework, Perthept, designed to uncover the latent robustness of ChatGPT by iteratively enhancing the level of reasoning. Comprehensive experiments suggest that Perthept demonstrates consistent performance in assisting ChatGPT in producing high-quality code summaries and maintaining robustness.
  - We conduct extensive experiments and ablation studies to evaluate the performance of Perthept on four datasets. Furthermore, we build a reasoning-required code datasets for future research by finely filtering the code samples from these well-studied datasets.

2 Robustness Issue of ChatGPT

The decline in ChatGPT’s robustness in code comment generation can be described as the following process. When a minor adversarial perturbation $\sigma$ is introduced into the original code $x$, the final input code $x + \sigma$ will dramatically degrade model’s performance [Zhou et al. 2022].

A high-quality code summary encompasses two critical aspects: the inclusion of a main purpose and information. However, minor adversarial perturbations often have a great impact on the loss of the main purpose in code summaries. Take the case in Figure 1 as an example, the term “unpack” is the main purpose, and the sentence “the compressed character translation table” is the information. From the perspective of code comprehension during software development, the term “decompress”, “unpack”, or “unzip” signifies the primary purpose of the code, which are the keywords for precise code comprehension. Therefore, the key to maintaining robustness for ChatGPT in code comment generation lies in not only accurately extracting the main purpose but also preserving sufficient information throughout the process.
3 Perthept

This study proposes a modular prompting framework Perthept to address the robustness challenges. The unique contribution of Perthept is maintaining robustness for ChatGPT in code comprehension, which is highlighted by successfully reasoning the main purpose of the code. Perthept consists of two key prompting paradigms, Chain-of-Structure and Reasoning-Enhancement. Chain-of-Structure preserves robust information from code structures, thus mitigating the potential negative effects resulting from unreliable semantics of code tokens. Reasoning-Enhancement paradigm focuses on improving the quality of generated code summaries by iteratively reasoning the main purpose of code while mitigating the inherent unreliability of ChatGPT’s responses.

3.1 Chain-of-Structure

The function of Chain-of-Structure is two-fold. First, decompose the complex programming language into natural language-based information to facilitate main purpose reasoning. Second, preserving reliable information by shifting the attention of ChatGPT from unreliable semantics of code tokens into reliable code structure information.

Accurately producing Chain-of-Structure with necessary information is crucial. The limited context space imposes two primary concerns in Chain-of-Structure prompting design: the length of outputs and the level of information detail. The prompting paradigm can be categorized into three different granularities, fine-grain, middle-grain, and coarse-grain, based on the level of detailed information that is incorporated.

Figure 3 shows how different granularities affect the information quality generated by ChatGPT. The fine-grained Chain-of-Structure focuses on the smallest units of the code, such as statements, expressions, variables, etc. This level of analysis provides detailed insights and precise understanding. However, it can also increase complexity and make the code analysis more intricate due to its lengthy generated text. The middle-grained Chain-of-Structure analyzes the code at a higher level of abstraction, considering functions, classes, modules, or components. This level provides a balance between fine-grain details and overall structure. The coarse-grained Chain-of-Structure analyzes the code at a high-level overview, such as architectural patterns, code design, or the overall flow of the program. This level of analysis focuses on understanding the large-scale structure and interactions between major components.

The ideal Chain-of-Structure should provide a clear description of code patterns and architectures since they form the foundation of code and represent its primary focus. It has been observed that both fine-grained and middle-grained Chain-of-Structure tend to prioritize detailed components such as expressions and variables while overlooking higher-level design patterns and architectural considerations. On the other hand, the coarse-grained Chain-of-Structure places more emphasis on these broader aspects. In this study, we adopt the few-shots learning approach and include several coarse-grained examples in the prompting pattern to generate Chain-of-Structure. Implementation details are included in the Appendix A.1.1.

3.2 Reasoning-Enhancement

Although Chain-of-Structure provides decomposes complex programming language information into simple natural language information, ChatGPT still struggles to effec-
The main purpose of the code is to generate a character array based on the input string by populating it with values and counts provided in a specific format within the string.

In summary, the code is responsible for decompressing a compressed string by expanding the count-value pairs into a character array.

Do you think this main purpose represent code’s intent? If not, go beyond this main purpose.

Reasoning-Enhancement

Figure 4: This figure provides an overview of the Reasoning-Enhancement process. Through multiple iterations of reasoning enhancement, Pertbeth is able to correct the main purpose from “generate a character array” to “decompressing a compressed string”.}


decompressing a compressed string by expanding the count-value pairs into a character array.

Do you think this main purpose represent code’s intent? If not, go beyond this main purpose.

Reasoning-Enhancement aims at leveraging the ability of ChatGPT to evaluate and refine its predictions to effectively enhance the depth of code comprehension, thus generating insightful and reliable code summaries. It is important to note that even with the same input, the predictions produced by ChatGPT cannot be guaranteed to remain consistent at all times, mainly due to the inherent unreliability of ChatGPT, which means ChatGPT is not guaranteed to catch the code’s main purpose all the time. To address this issue, we design the Reasoning-Enhancement prompting to be iterative for continuously enhancing the reasoning level. As a result, this approach effectively mitigates the potential negative impact caused by such unreliability.

Reasoning-Enhancement is a self-refinement prompting that prompts ChatGPT to improve its outputs, and it consists of two prompting components: self-evaluation and further-enhancement. For self-evaluation, we use the trigger sentence “Do you think this main purpose represents code’s intent?” that enables ChatGPT to evaluate its reasoning results. For further-enhancement, we adopt “if not, go beyond this main purpose for a higher-level abstraction” to ensure that each subsequent prediction during iteration is at a higher level of reasoning. These two prompting components are integrated together and iteratively prompting ChatGPT to perform higher-level reasoning, resulting in the production of reliable code summaries and effectively mitigating the negative impact of the inherent unreliability of ChatGPT’s responses.

As observed in the final code summary in Figure 4, ChatGPT successfully corrects the main purpose from “generate a character array” to “decompressing a compressed string”. More implementation details are included in the Appendix.

4 Experiments

4.1 Setups

Datasets. We evaluate our workflow on four publicly available datasets and one filtered datasets collected on our own, respectively. The statistics of each datasets are shown in Table 2 in Appendix B.1, and the algorithm of reasoning-required code samples collection is included in Appendix B.2.

Adversarial Transformations. We follow [Sontakke et al. 2022] to use javalang and ast packages to programmatically transform the codes, which includes: (a) Replacing meaningful user-defined function names with more generic function names (e.g., func). (b) Replacing meaningful user-defined local variable names with more generic variable names (e.g., var1, var2, et al) such that data dependencies are preserved.

Ground Truth Refine. To ensure a fair comparison between NLP and prompting baselines, we refine the ground truth by combining high-level abstraction summaries and brief descriptions. More information is provided in Appendix A.1.2.

Baselines. We consider the following prompting paradigms and NLP models: (a) Direct Prompt: a standard zero-shot paradigm that prompts an LLM to directly solve tasks or answer questions. (b) Chain-of-Thought [Kojima et al. 2022]: a prompting paradigm that engages LLMs into “think step by step” reasoning. (c) SELF-REFINE [Madaan et al. 2023]: a prompting paradigm that improves initial outputs from LLMs through iterative feedback and refinement. (d) Code-BERT [Feng et al. 2020]: it is pre-trained on vast bimodal corpora with the masked language model and replaced token detection. (e) UniXcoder [Guo et al. 2022]: it is a pre-trained model that utilizes multi-modal data for better code fragment representation through contrastive learning.

Evaluation Metrics. Common performance metrics such as character-level Precision and F1 score are employed in our experiments. Moreover, a ChatGPT-based scorer is used to measure the semantic Accuracy of answers (e.g., “Get” and “Retrieve” are semantically close but cannot be identified by Precision and F1). More information is provided in Appendix A.4.

4.2 Results and Discussions
Perthept excels over all baselines consistently. Table 1 shows the main evaluation results on public datasets based on two popular encoder-decoder models and gpt-3.5-turbo. For the prompting paradigm, we test each code sample twice on two popular encoder-decoder models and gpt-3.5-turbo. For robust training approach for NLP models, as well as the robustness compared to NLP baselines. This is due to the less perturbations and can maintain consistent performance. Among prompting paradigms, we observe the sheer win of Perthept. The baseline prompting approaches manage to maintain relatively low drops in Precision and F1 scores compared to NLP baselines due to ChatGPT’s proficiency in generation tasks. However, their Accuracy drops are comparatively higher, indicating a higher incidence of incorrect purpose reasoning, particularly when dealing with complex code scenarios that require main purpose reasoning.

It is worth noting that the majority of code samples in public datasets are relatively simple, and intricate main-purpose reasoning is not often required. However, the extensive volume of data can still provide a rough assessment of the performance of each prompting paradigm, even though the differences may not be substantial. In contrast to baseline prompt-
ing paradigms, Perthept achieves higher scores for less robust transformed codes while experiencing lower score drops, indicating that it not only maintains a high level of robustness by successfully abstracting the main purpose of complex code samples but also ensures a high quality of code summaries at the same time. During the evaluation on our Reasoning-required datasets, Perthept demonstrates superior performance compared to baseline approaches. It achieves high scores while maintaining low score drops, indicating the effectiveness of main purpose reasoning in robustness improvement for code comprehension.

Why Perthept outperforms all prompting baselines in preserving robustness. It is observed that the baseline prompting paradigm lacks clear guidance for reasoning. When utilizing the same baseline approach to prompt ChatGPT multiple times, ChatGPT tends to modify the way it describes the same concept at the same reasoning depth in each response, instead of offering higher-level insights. In contrast, with the guidance of main purpose reasoning, Perthept systematically enhances the depth of reasoning to provide deeper insights, which is the key approach to outperforming all other prompting baselines. We conduct a side experiment where we remove the guidance keywords “with code’s main purpose” from Perthept prompting. We observed a substantial drop in the performance of Perthept, and the reasoning level in each iteration remained unchanged. This experiment highlights the significance of guidance keywords in the task of code comment generation.

Perthept is relatively robust upon ChatGPT’s unreliability. Given the crucial semantic information provided by function and variable names, ChatGPT does not consistently generate correct code summaries due to its unreliability. The unreliability of ChatGPT can be characterized by the fact that it generates different code summaries for the same input code and prompts on each iteration. This explains the score discrepancy among the prompting paradigms when evaluating Accuracy scores on original codes. Take direct prompt as an example, ChatGPT occasionally generates incorrect code summaries by providing verbose and redundant descriptions instead of accurately extracting the main purpose of the code. This behavior could be attributed to the lack of clear guidance in the prompting paradigm, which results in ChatGPT not consistently prioritizing the main purpose. This issue also affects Chain-of-Thought and SELF-REFINE prompting paradigms. On the contrary, the design of Perthept is based on purpose-reasoning which effectively mitigates the impact of unreliability by progressively enhancing the code’s main purpose at a higher-level abstraction during each iteration. Through observation, ChatGPT maintains the same response when it is unable to provide a higher-level abstraction, rather than producing incorrect code summaries, thus preserving the robustness. A more specific case study is provided in Appendix A.3.

4.3 Ablation Study

Without Reasoning-Enhancement prompting. The Reasoning-Enhancement module plays a crucial role in accurately abstracting the main purpose of the code. As depicted by the orange bar in Figure 5, the removal of the Reasoning-Enhancement module results in an Accuracy drop on all benchmarks, indicating that the Reasoning-Enhancement module effectively improves the robustness of complex code comprehension for ChatGPT.

Without Chain-of-Structure prompting. For accuracy scores, although Chain-of-Structure may not preserve the higher-level reasoning capabilities, it plays a crucial role in preserving important information, which is vital for generating high-quality code comments. Additionally, the architectural design of Chain-of-Structure follows the chain-of-thought prompting paradigm, breaking down complex programming language constructs into the step-by-step analysis. Therefore, this approach inherently incorporates some degree of reasoning, which helps explain why Perthept with only Chain-of-Structure can achieve higher performance compared to Perthept with only the direct summary paradigm of “summarize the code with code’s main purpose”. Furthermore, based on our observations, if we do not enhance the code’s main purpose during the prompting process, there is a risk of generating incorrect summarized main purposes, which does not provide insights into code summary but makes code summary much longer. This can have a detrimental effect on the overall quality of the code summary when testing on a large number of code samples. For precision and F1 scores, the removal of Chain-of-Structure leads to a substantial drop in the scores, indicating that the function of Chain-of-Structure is more focused on information preservation rather than reasoning.

K value for Reasoning-Enhancement paradigm. Another input of Perthept is a user-defined value $K$, representing the number of Reasoning-Enhancement paradigm included in Perthept. For instance, 0 represents no such paradigm, and 2 illustrates that we repeat the Reasoning-Enhancement paradigm once. As depicted by Figure 6, the performance is primarily influenced by the presence or absence of the Reasoning-Enhancement paradigm, while the number of repetitions has a limited impact on accuracy improvement. However, to mitigate the unreliability of ChatGPT generations, we finally choose 3 as the K value in this study (i.e., repeat twice).

Additional Observations Upon analyzing the self-feedback generated by ChatGPT, we discover that the suggestions provided by SELF-REFINE prompting rarely focus on providing deeper insights or higher-level reasoning in each self-refine process. Due to the influence of unrelated self-refine suggestions, the refined code summary not only suffers from information loss but also deviates from the main purpose of the code. As a result, the overall generation quality becomes worse.

5 Related Work

Prior to the emergence of extremely large language models (e.g., ChatGPT, GPT4, et al), the most widely adopted approach for code comment generation involves training [Hu et al. 2018a, Wei et al. 2019] and fine-tuning [Feng et al. 2020, Ahmad et al. 2021, Guo et al. 2022] specific encoder-decoder
models. Among these works, several advancements, including incorporating various attention mechanisms, have demonstrated state-of-the-art performance in generating accurate code comments [Ahmad et al. 2020, Wu et al. 2020]. However, these state-of-the-art models both face robustness challenges, as they exhibit poor performance when subjected to even minor perturbations in the input code [Sontakke et al. 2022]. To mitigate the negative impact of minor perturbations, adversarial training [Goodfellow et al. 2014] has been widely adopted to enhance model performance [Zhou et al. 2022, Jha and Reddy 2022]. However, due to the limited generalization of the model itself due to small training datasets, these improved models still encounter robustness challenges when presented with new code patterns.

With the rapid development of LLMs, the code comment generation approach is evolving to prompt LLMs. ChatGPT is a recent chatbot service released by OpenAI [Wang et al. 2023] and has shown great potential in generating coherent text, which can be leveraged for assisting code comprehension during development. However, ChatGPT still faces robustness challenges in consistently producing high-quality code summaries because it relies heavily on the unreliable semantics of code tokens, which can be easily altered by introducing even minor perturbations, resulting in generating useless and lengthy descriptions of code.

As fine-tuning ChatGPT is costly, prompting design emerges as a practical solution to address the robustness challenges. A thoughtfully designed prompt can significantly impact ChatGPT’s predictions. Recent studies have shown that the utilization of reasoning prompts can significantly enhance the accuracy of predictions. Recent popular prompting approaches are optimized to allow a single large language model to better perform a variety of tasks [Li and Liang 2021, Lester et al. 2021, Reif et al. 2021]. Among these works, [Wei et al. 2022] proposes to apply chain-of-thought prompts to improve the ability of large language models to perform complex reasoning to solve complex tasks. Nevertheless, neither of these works demonstrates a direct impact on addressing the robustness issues for LLMs to improve code comprehension.

In contrast to previous studies, our research specifically aims to activate ChatGPT’s purpose-reasoning capability to enhance code comprehension while mitigating the adverse impact caused by robustness perturbations.

6 Conclusion

We present Perthept, a modular prompting framework to solve robustness issues efficiently in code comprehension for ChatGPT by leveraging main-purpose reasoning guidance and iterative reasoning enhancement. With clear reasoning guidance and iterative higher-level reasoning enhancement, Perthept effectively maintains high levels of robustness in complex code comprehension tasks. Comprehensive experiments reveal superior performance of Perthept in achieving a high level of robustness while simultaneously ensuring high-quality code comment generation. A side study also shows that Perthept has relatively robust performance under the unreliability of ChatGPT generations. Future improvements beyond Perthept involve prompting optimization (e.g., token reduction), robustness improvement for dynamic code comprehension, modular LLM fine-tuning, and augmentation with tools.
Acknowledgements

We thank the anonymous reviewers for their valuable comments. This work is partially funded by NSF 2050007 and a Google gift.

References


[Hu et al., 2018b] Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. Summarizing source code with transferred api knowledge. 2018.


