



# Developers' Perspective on Trustworthiness of Code Generated by ChatGPT: Insights from Interviews

Zeinab Sadat Rabani<sup>1</sup>(✉), Hanieh Khorashadizadeh<sup>1</sup>, Shirin Abdollahzade<sup>2</sup>,  
Sven Groppe<sup>1</sup>, and Javad Ghofrani<sup>1</sup>

<sup>1</sup> Universität zu Lübeck, Lübeck, Germany  
[zeinab.rabani@uni-luebeck.de](mailto:zeinab.rabani@uni-luebeck.de)

<sup>2</sup> University of Guilan, Rasht, Iran

**Abstract.** The emergence of ChatGPT as a tool for code generation has garnered significant attention from software developers. Nevertheless, the reliability of code produced by Large Language Models (LLMs) like ChatGPT remains insufficiently explored. This article delves into the realm of ChatGPT-generated code, aiming to investigate the perspectives of esteemed programmers and researchers through interviews. The consensus among interviewees highlights that code generated by ChatGPT often lack accuracy, necessitating manual debugging and substantial time investment, particularly when dealing with complex code structures. Through a comprehensive analysis of the interview findings, this article identifies five primary challenges inherent to ChatGPT's code generation process. The core objective of this research is to engage in an exploration of ChatGPT's code generation trustworthiness, drawing insights from interviews with experts. By facilitating insightful discussions, the research aims to pave the way for proposing impactful enhancements that bolster the reliability of ChatGPT's code outputs. To enhance the performance and overall dependability of LLMs, the article presents seven potential solutions tailored to address these challenges.

**Keywords:** Large Language Model (LLM) · ChatGPT · Trustworthiness · Code Generation

## 1 Introduction

OpenAI has developed ChatGPT, a chatbot based on a large language model (LLM) tailored for conversational purposes. OpenAI is improving the LLM continuously. It has been started with the first version of this model which is called Generative Pre-trained Transformer(GPT) in 2018 [15]. ChatGPT is using version 3.5 and 4 of GPT while version 5 is going to release in the near future. By leveraging its comprehension of the context and conversation history, ChatGPT is capable of generating responses that closely resemble those of a human [11].

ChatGPT has gained significant global attention and is widely discussed and utilized across various domains and applications [3]—particularly programmers

who find it valuable for code generation and error identification purposes [20]. ChatGPT has been utilized for various purposes in code generation, including code writing and debugging, preparing for programming interviews, working on programming-related assignments, and other related tasks [4]. Various other AI-based code generation tools, such as Amazon’s CodeWhisperer and Google’s Bard, are also available. Table 1 presents some of the existing code generation AI tools.

**Table 1.** Existing Code Generation AI Tools

|                      | Tools                         |
|----------------------|-------------------------------|
| OpenAI               | ChatGPT, OpenAI Codex         |
| Google               | BARD                          |
| Tabnine              | Tabnine                       |
| CodeT5               | Salesforce                    |
| GitHub Copilot       | GitHub, OpenAI, and Microsoft |
| Amazon CodeWhisperer | Amazon                        |

Many developers commonly reuse code they find in open-access repositories like GitHub, online forums, and websites such as Stack Overflow. Large Language Models (LLMs) are now being used by software developers to speed up their projects by generating code. However, it’s important to be cautious about the quality of code generated by LLMs or other third-party sources, even if it seems to be free from major bugs or vulnerabilities in experimental situations. [11]. Copying code snippets from such sources can introduce security vulnerabilities, compatibility issues, and make project maintenance more challenging. Several studies, tools, and projects have delved into the reliability of reused code from third-party sources and its impact on project development and maintenance. It’s worth noting that ChatGPT has its limitations, particularly in terms of the accuracy and trustworthiness of the code it generates. Occasionally, the model may produce responses that sound plausible but are factually incorrect or nonsensical, a phenomenon referred to as “hallucination.” [19]. It is also sensitive to input phrasing and can exhibit biases present in the training data. For this reason, reusing the generated code by ChatGPT can cause security or maintenance issues in software projects. If we assume that the generated code is correct and explainable, then there is no guarantee that this code will not cause problems in the future of the project. This issue which is addressed as the trustworthiness of generated code by ChatGPT—as well as other LLM-based models—is overlooked amid the excitement over ChatGPT’s capabilities.

In this paper, we address the problem of the trustworthiness of generated code by ChatGPT. In software development, “trustworthiness” refers to the quality of a software system or application in terms of its reliability, security, and overall performance. Trustworthiness encompasses several key aspects such as security, stability, or performance. Achieving trustworthiness often involves

rigorous testing, security assessments, adherence to coding standards, and ongoing maintenance and updates to address emerging issues. Our contribution is to provide an overview of challenges and opportunities for starting the research on issues related to this topic. Because the subject is relatively new, there is a limited body of research and resources addressing the challenges associated with reusing the code generated by ChatGPT. As a result, we initiated our investigation of this field by conducting a study based on interviews. We interviewed 20 developers who use ChatGPT for code generation and asked their opinion about trusting the generated code of ChatGPT. We analyzed the results of the interviews with the aim of examining the strengths and weaknesses of ChatGPT in code generation. Drawing from their expertise, we have identified various challenges and provided recommendations to mitigate potential issues regarding the trustworthiness of the generated code of ChatGPT.

The results of our interview-based study can be used as a starting point for further studies and development of methods and tools regarding automated controlling and checking the trust issues in generated code with LLMs. Furthermore, this study fills the gap of knowledge and communication between software developers who are using the development tools and the machine learning experts who are developing tools like ChatGPT.

The rest of this paper is structured as follows. In Sect. 2, we review prior research carried out in this domain. In Sect. 3 we delve into the interviews and the concerns raised by the interviewees. In Sect. 4 we discuss potential approaches that can improve the reliability and authenticity of ChatGPT.

## 2 Related Work

Within this domain, we delve into two distinct contexts. Firstly, we discuss articles related to LLMs and mostly ChatGPT, highlighting their relevance. Secondly, we explore articles that focus on the verification of reliable coding, providing insights into this particular area.

### 2.1 ChatGPT

While ChatGPT demonstrates impressive language generation capabilities, it is important to note that it has limitations. The model can sometimes produce responses that may be plausible-sounding but factually incorrect or nonsensical called hallucination [19]. Numerous techniques exist for addressing hallucination within the field of Natural Language Generation (NLG). One strategy involves considering the degree of hallucination as a manageable characteristic and restraining it to a minimum using controlled generation methods like controlled re-sampling and control codes, which can either be manually input or predicted automatically. Another method entails employing metrics to assess the excellence of the produced text, including factors like factual coherence. Additionally, refining pre-trained language models using synthetic data containing

automatically integrated hallucinations is proposed as an alternative technique for pinpointing hallucinatory elements in summaries [8].

Sun et al. [18] proposed an evaluation of ChatGPT's performance in code summarization using three metrics: BLEU, METEOR, and ROUGE-L. BLEU measures how much a generated summary is like the provided reference summaries. It looks at the words used and higher scores mean the generated summary is more similar to the references. METEOR also compares the generated summary to the reference summaries, but it looks at many aspects like matching words, and it gives higher scores when the generated summary is more like the references. ROUGE-L checks the longest shared part between the generated summary and the references. Higher scores mean the generated summary and references have more words in common. These metrics gauge the quality of the generated comments in comparison to the ground-truth comments. Among these metrics, ChatGPT's performance is inferior to three state-of-the-art models (NCS, CodeBERT, and CodeT5) in terms of BLEU, METEOR, and ROUGE-L scores. However, it is noteworthy that ChatGPT achieves a higher METEOR score compared to the other models, implying that its generated comments may possess specific linguistic qualities captured by METEOR but not entirely reflected in the other metrics [18]. Despite this, recent studies have revealed limitations in match-based metrics when evaluating code. For example, [16] discovered that BLEU struggles to capture code-specific semantic features and proposed various semantic adjustments to improve the scoring accuracy [1]. As a result recent studies have shifted their focus to prioritize functional correctness. Under this approach, a code sample is deemed correct only when it successfully passes a predefined set of unit tests [1]. In their assessment, Kul [10] utilized the pass@k metric to assess functional correctness. They produced k-code samples for each problem and deemed a problem solved if any of the samples generated passed the unit tests. The reported metric reflects the overall fraction of successfully solved problems using this particular criterion.

Liu et al. [11] state that the effectiveness of ChatGPT in generating code is often influenced by how prompts are designed. Two main factors affect prompt design:

1. Chain-of-Thought (CoT) prompting: This strategy allows an LLM to solve problems by guiding it to produce a sequence of intermediate steps before providing the final answer. CoT prompting has been extensively studied and applied to guide ChatGPT in code generation tasks.
2. Manual Prompt Design: Prompt design and multi-step optimizations are based on human understanding and observations. The knowledge and expertise of the designer can impact the performance of the prompts used. To minimize bias, prompt design, and combination choices should be based on a large and randomized sample size.

To enhance generation performance through prompt design, they propose a two-step approach:

1. Prompt Description: Analyze the requirements of a code generation task and create a basic prompt in a natural manner. Then, present the basic prompt

to ChatGPT and ask for suggestions on how to improve it. Incorporate ChatGPT's suggestions to refine the prompt further.

2. **Multi-Step Optimizations:** Evaluate the prompt designed in the first step using samples from the training data of the relevant dataset. Analyze the generation performance by comparing it with the ground-truth results. Continuously optimize the generation results by providing ChatGPT with a series of new prompts.

It is important to acknowledge that the prompt design and combination choices were based on a limited number of tests. Conducting more tests can contribute to further enhancing the effectiveness of the designed prompts and reinforcing the conclusions and findings [11].

Jansen et al. [7] assert that large language models (LLMs) have the potential to tackle certain issues related to survey research, such as question formulation and response bias, by generating responses to survey items. However, LLMs have limitations in terms of addressing sampling and nonresponse bias in survey research. Consequently, it is necessary to combine LLMs with other methods and approaches to maximize the effectiveness of survey research. By adopting careful and nuanced approaches to their development and utilization, LLMs can be employed responsibly and advantageously while mitigating potential risks [7].

Feng et al. [4] introduces a crowdsourcing data-driven framework that integrates multiple social media data sources to assess the code generation performance of ChatGPT, a generative language model. The framework consists of three main components: keyword expansion, data collection, and data analytics. The authors utilized topic modeling and expert knowledge to identify programming-related keywords specifically relevant to ChatGPT, thus expanding its initial seed keyword. With these expanded keywords, they collected 316K tweets and 3.2K Reddit posts discussing ChatGPT's code generation between December 1, 2022, and January 31, 2023. The study discovered that Python was the most widely used programming language in ChatGPT, and users primarily employed it for tasks such as generating code snippets, debugging, and producing code for machine learning models. The study also analyzed the temporal distribution of discussions related to ChatGPT's code generation and found that peak activity occurred in mid-January 2023. Furthermore, the research examined stakeholders' perspectives on ChatGPT's code generation, the quality of the generated code, and the presence of any ethical concerns associated with the generated code [4].

## 2.2 Trustworthiness

Dener and Batisti [2] discuss the security and ethical considerations surrounding the use of LLMs such as ChatGPT. While artificial intelligence (AI) advancements in natural language processing (NLP) have been remarkable, there is a growing apprehension regarding the potential safety and security risks and ethical implications associated with these models. The article points out that ChatGPT's filters are not completely foolproof and can be circumvented through

creative instructions and role-playing. Utilizing large language models like ChatGPT raises ethical concerns related to privacy, security, fairness, and the possibility of generating inappropriate or harmful content. The paper emphasizes the necessity for further research to address the ethical and security implications of large language models. It also presents a qualitative analysis of ChatGPT's security implications and explores potential strategies to mitigate these risks. The intention of the paper is to provide insights to researchers, policymakers, and industry professionals about the complex security challenges posed by LLMs like ChatGPT. Additionally, the paper includes an empirical study that assesses the effectiveness of ChatGPT's content filters and identifies potential methods to bypass them. The study demonstrates the existence of ethical and security risks in LLMs, even when protective measures are implemented. In summary, the paper underscores the ongoing need for research and development efforts to address the ethical and security implications associated with large language models like ChatGPT [2].

Myre and Robinson [13] assert that the AI Trust Framework and Maturity Model (AI-TFMM) is a method aimed at enhancing the measurement of trust in AI technologies used by Autonomous Human Machine Teams & Systems (A-HMT-S). The framework addresses important aspects like security, privacy, explainability, transparency, and other ethical requirements for the development and implementation of AI technologies. The maturity model framework employed in this approach helps quantify trust and associated evaluation metrics while identifying areas that need improvement. It offers a structured assessment of an organization's current trust in AI technologies and provides a roadmap for enhancing that trust. By utilizing the maturity model framework, organizations can track their progress and continually improve trust in AI technologies. Overall, the maturity model framework is an invaluable tool for quantifying trust in AI technologies and ensuring ethical practices in their development and use. The AI-TFMM is also examined in relation to a popular AI technology, ChatGPT, and the article concludes with results and findings from testing the framework. The AI-TFMM serves as a critical framework in addressing key questions about trust in AI technology. Striking the right balance between performance, governance, and ethics is essential, and the AI-TFMM provides a method for measuring trust and associated evaluation metrics. The framework can be utilized to enhance the accuracy, efficacy, application, and methodology of the AI-TFMM. The article also identifies areas for future research to fill gaps and improve the AI-TFMM and its application to AI technology. In summary, the AI Trust Framework and Maturity Model is a valuable tool for enhancing trust in AI technologies and ensuring their ethical development and application [13].

Ghofrani et al. [5] conducted a study focusing on trust-related challenges when reusing open-source software from a human factors perspective. The objective of the study was to gain insights into the concerns of developers that hinder trust and present potential solutions suggested by developers to enhance trust levels. Sixteen software developers with 5 to 10 years of industry experience participated in exploratory interviews for the study. The findings indicate that

developers possess a good understanding of the associated risks and have a reasonable level of trust in third-party open-source projects and libraries. However, the proposed solutions generally lack recommendations for utilizing automated tools or systematic methods. Instead, the suggested solutions primarily rely on developers' personal experiences rather than existing frameworks or tools. The study concludes that these limitations are interconnected, and the absence of ongoing support can gradually lead to security vulnerabilities. In turn, a project with numerous vulnerabilities may become prohibitively expensive to maintain, ultimately resulting in abandonment [5].

Wermke et al. [21] explore the advantages and difficulties associated with integrating open-source components (OSCs) into software projects. The research reveals that OSCs have a significant role in numerous software projects, with most projects implementing company policies or best practices for including external code. However, the inclusion of OSCs presents specific security challenges and potential vulnerabilities, such as the use of code contributed by individuals without proper vetting and the obligation to assess and address vulnerabilities in external components. The study also indicates that many developers desire additional resources, dedicated teams, or tools to effectively evaluate the components they include. Additionally, the article outlines the study's approach, which involved conducting 25 detailed, semi-structured interviews with software developers, architects, and engineers involved in various industrial projects. The participants represented a diverse range of projects and backgrounds, spanning from web applications to scientific computing frameworks. Overall, the findings emphasize the importance of thoroughly considering the security implications of incorporating OSCs into software projects and the necessity for companies to establish policies and best practices to mitigate the impact of vulnerabilities in external components [21].

In contrast to above mentioned contributions, we focus on trustworthiness in generated code by LLMs in particular ChatGPT in the practical stage. The focus of the previous studies are mainly on security concerns of code from third parties which are written by a human or generated from Models which are created by human. This kind of code is different than written code by LLM-based tools. Therefore, it is necessary to investigate the measures which may be required for trust in the code generated with new LLM-based tools.

### 3 Methodology

Because our research topic has not been extensively explored, interview-based studies align effectively with our chosen research methodology. This approach allows us to thoroughly investigate the subject and gather valuable insights due to its suitability for exploring uncharted areas of inquiry. Consequently, among the spectrum of available research methods, interviews are selectively employed to elicit a more comprehensive array of viewpoints pertaining to the research subject, thereby serving as a foundational step for the progression of future investigations in this domain. [12]. In this context, we distributed a set

of interview questions to programmers with the aim of collecting their insights and viewpoints regarding the generation of code using ChatGPT. To fulfill this objective, we formulated interview questions specifically tailored for individuals with diverse expertise in this particular. We conducted interviews with 20 individuals, including developers, engineers, students, and researchers, who work in reputable companies with large teams. As stated by Saunders and Townsend [17], the most common sample size in qualitative research typically falls between 15 and 60 participants. To gather their perspectives, we posed a set of 10 questions that delved into their positions, programming expertise, frequency of ChatGPT usage, reliability of the generated code, and suggestions for future improvements. Our Interviews are inspired by a questionnaire from Parnel et al. [14]. Here are the questions:

- What is your expertise?
- What is your current activity?
- How big is your working company?
- Do you employ generative AI models, like ChatGPT, for the purpose of code generation? Furthermore, what is the frequency of utilization for code produced by ChatGPT, and in which typical scenarios does this usage occur?
- Can you recall a point in this situation when you did not trust the generated code by ChatGPT?
- Would you generally tend to trust the generated code by ChatGPT? Please expand on why.
- Do you have any distrust in the generated code by ChatGPT? What is the cause of this distrust? How could it be repaired?
- Would you have any reason not to trust the generated code by ChatGPT in the future?
- For the future, how reliable/dependent do you view the generated code by ChatGPT to be?
- What information/knowledge would you need to trust the generated code by ChatGPT?

Table 2 lists the summary of the various backgrounds and expertise of the interviewees.

**Table 2.** Overviewing of the interviewees’ expertise

| Expertise             | Count |
|-----------------------|-------|
| Frontend Engineer     | 6     |
| Software Engineer     | 4     |
| Student               | 4     |
| Android/iOS Developer | 3     |
| Backend Developer     | 2     |
| Data Scientist        | 1     |



The majority of participants reported using ChatGPT on a daily basis, with some utilizing tools like the GitHub Co-pilot plugin and Tabnine. However, all interviewees shared instances where the generated code proved to be unreliable and inaccurate, necessitating manual verification and extensive time investment. Some participants mentioned using ChatGPT-generated code as a starting point, but modifying it extensively to suit their specific use cases. Overall, the consensus among the interviewees was that the generated code should not be blindly trusted or relied upon, and thorough checking and review are essential. Fig. 1 shows one instance provided by interviewees highlighting the unreliability of ChatGPT. He first asked ChatGPT to write code in Python to implement a box plot for a dataframe. The output worked fine, but on the second step, he asked ChatGPT

HA I want to have different colors for each quartile



Certainly! You can achieve this by specifying the `boxprops` parameter in `boxplot` function of `matplotlib`. This allows you to customize various properties of the boxes, including their colors. Here's an example of how to create a box plot with different colors for each quartile using Matplotlib:

```
python Copy code

import pandas as pd
import matplotlib.pyplot as plt

# Sample DataFrame
data = {
    'A': [10, 15, 20, 25, 30, 35, 40, 45, 50],
    'B': [5, 12, 18, 24, 31, 37, 42, 47, 52].
}

df = pd.DataFrame(data)

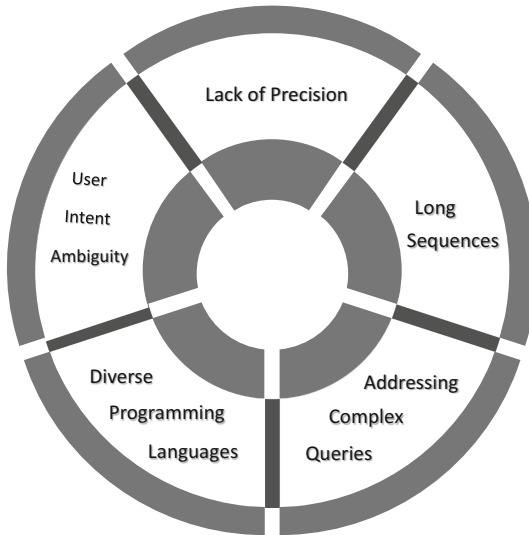
# Define colors for each quartile
box_colors = ['skyblue', 'lightgreen', 'lightcoral', 'gold']

# Creating a box plot with custom colors for each quartile
plt.boxplot(df.values, labels=df.columns, boxprops=dict(facecolor=box_colors))
plt.title('Box Plot with Custom Quartile Colors')
plt.ylabel('Values')
plt.show()
```

Fig. 1. Asking ChatGPT to provide different colors for each quartile in a box plot

to provide that box plot with different colors for each quartile which is depicted in Fig. 1 where ChatGPT provided some code that resulted in errors. The error states that `Line2D.set()` got an unexpected keyword argument 'facecolor'. So ChatGPT might hallucinate when it has no idea what to produce as output [9].

It's important to highlight that all interviewees have experience with ChatGPT 3.5, but it is important to recognize the advancements made in the updated model, ChatGPT 4.0. This newer version has made strides in addressing some of the issues and bugs that were present in the initial release. However, it is pertinent to acknowledge that accessibility to ChatGPT 4.0 is limited, as it is not available for free. This limitation poses a challenge for certain students and programmers who may lack the financial resources to afford the updated version.



**Fig. 2.** Main problems of ChatGPT for generating reusable code according to our interviews

After analyzing these questions, we identified five issues that were unanimously agreed upon by all interviewees. These problems undermine the trustworthiness of ChatGPT and its ability to provide accurate answers. Figure 2 illustrates these problems briefly. The full description of each problem is as follows:

1. **Lack of Precision:** ChatGPT may not generate code that is as precise and efficient as what experienced developers can produce. Code requires strict adherence to syntax, logic, and optimization, which might not always be guaranteed by the model.
2. **Long Sequences:** ChatGPT has a token limit that can restrict the length and complexity of code generated in a single prompt. This can limit the

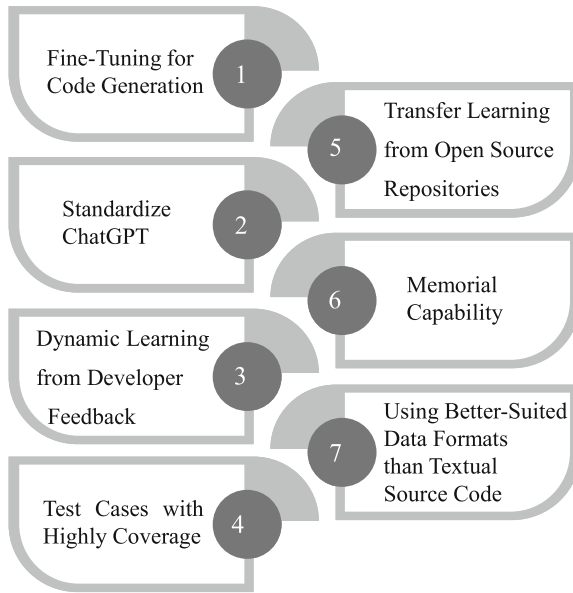
model's ability to generate longer or more complex code segments. This might necessitate breaking down the task into smaller parts.

3. **User Intent Ambiguity:** ChatGPT might misinterpret user intent, leading to code that doesn't accurately address the user's needs.
4. **Diverse Programming Languages, Libraries, and Frameworks:** The realm of programming encompasses an extensive spectrum of languages tailored to diverse applications, ranging from versatile options like Python, Java, and C++ to specialized alternatives such as R and MATLAB. However, it is imperative to acknowledge that while ChatGPT has been extensively trained on a substantial corpus of internet text, its mastery is not all-encompassing. Its competence across programming languages is not uniform, exhibiting variations, and certain languages may command a more limited familiarity. Additionally, its grasp may be less robust concerning nascent or niche programming languages. Thus, it is advisable to provide contextual specifics when dealing with relatively uncommon or specialized technologies, as ChatGPT's exposure during training might not encompass these facets comprehensively. Also, many projects rely on specific libraries or frameworks. The model might not be up-to-date on these tools, causing it to generate code that doesn't effectively leverage available resources.
5. **Addressing Complex Queries:** The queries presented to ChatGPT are intricate, spanning from inquiries about solving theoretical mathematical concepts to abstract ideas that even experts might find challenging. It's crucial to recognize that ChatGPT's training and intended purpose are distinct, which limits the range of questions it can adeptly address.

## 4 Future Research Directions

Considering the issues and shortcomings discussed earlier, we can suggest potential solutions and future research to enhance the performance of ChatGPT and formulate a roadmap and policy for its continuous development and improvement. Figure 3 presents a visual representation of recommendations aimed at enhancing the performance of LLMs in code generation.

1. **Fine-Tuning for Code Generation:** Tailoring ChatGPT's training on specific programming domains and languages could enhance its code generation accuracy within those areas, addressing challenges related to syntax and semantics.
2. **Standardize ChatGPT:** To ensure clarity and facilitate effective usage of ChatGPT, it is important to establish and publish documents and standards that outline guidelines for its utilization. Currently, there is a lack of standardized documentation that clearly defines how to use ChatGPT and sets expectations regarding its capabilities and limitations. By creating comprehensive documentation, developers and users would have access to a clear set of instructions, best practices, and recommended approaches for interacting with ChatGPT. This would encompass guidance on various aspects, such as



**Fig. 3.** Suggestion for enriching LLMs for generating code

input format, understanding response types, addressing potential biases, and providing context to optimize the quality of responses. The availability of such documents and standards would promote consistent and informed utilization of ChatGPT, enabling users to leverage its potential effectively and navigate its capabilities with confidence.

3. **Dynamic Learning from Developer Feedback:** Developing methods for ChatGPT to learn from developers' corrections and suggestions, allowing the model to continuously improve its code generation abilities.
4. **Test Cases with High Coverage:** To validate the quality of the generated code, it is imperative to develop exhaustive test cases that achieve high coverage. These test cases play a crucial role in confirming the functionality and accuracy of the code produced by ChatGPT. By formulating test cases that encompass a diverse range of scenarios, including edge cases and potential inputs, we can thoroughly scrutinize the reliability and precision of the generated code. Robust test coverage ensures a comprehensive examination of various code aspects, including corner cases and potential pitfalls. This scrutiny allows us to detect any inconsistencies or errors within the generated code and implement corrective measures as needed. Through meticulous testing, we can establish a high level of confidence in the authenticity and effectiveness of the generated code, thus validating its applicability to real-world contexts

5. Transfer Learning from Open Source Repositories: Exploring the use of open-source code repositories to train the model on real-world coding patterns and practices, improving its understanding of actual coding conventions.
6. Memorial Capability: The suggestion is to enhance the ChatGPT model by enabling it to remember previous searches and provide tailored solutions based on previous questions and the user's skill level. For example, if someone asks multiple programming-related questions involving various libraries and frameworks, the response should include generated code incorporating sophisticated algorithms and a broader perspective.
7. Using Better-Suited Data Formats than Textual Source Code: Contributions like [6] show that textual source code might not be the best choice for the training of machine learning tasks on source code. Actually in the experiments of [6], higher accuracies with less memory consumption are achieved for the same amount of training data when eliminating variable names but still maintaining declaration-usage relations in a graph representation of the source code. Future Large Language Models may be based on these or similar techniques for the purpose of generating better code with less training data.

## 5 Discussion

The purpose of the interview is to gather valuable insights and opinions of prominent programmers and researchers in relation to the code generated by ChatGPT. Great care was taken in the selection process to ensure a comprehensive representation of diverse perspectives, avoiding any potential biases towards specific individuals. Although the number of participants in the interviews may not have been extensive, their feedback still provides significant and meaningful information about the current state of LLMs and especially ChatGPT. Through these interviews, various problems and weaknesses of the model have been brought to light, shedding light on areas that require improvement.

Nonetheless, the research conducted through these interviews serves as a significant starting point for initiating discussions within the researchers' community. The aim is to explore the reliability and trustworthiness of ChatGPT and brainstorm potential solutions to enhance the model. By fostering a collaborative environment for knowledge sharing and problem-solving, this research endeavor hopes to contribute to the ongoing efforts to improve LLMs and ensure its effectiveness and usability for a wider range of users.

## 6 Conclusion

This article delves into a topic that has sparked considerable debate: trustworthiness of the code generated by ChatGPT. By conducting interviews with esteemed programmers and researchers, the authors aim to delve into their perspectives regarding the trustworthiness of the code produced by the model. The consensus emerging from these interviews is that ChatGPT's generated code frequently

lacks precision, necessitating manual debugging, and substantial time investment, particularly when handling complex code scenarios. Through a meticulous analysis of the interview data, the article identifies five key challenges inherent to ChatGPT's code generation process. These challenges encompass issues related to the accuracy, code length limitations of a single prompt, interpreting user intent, accommodating diverse programming languages, tools, and methodologies, as well as addressing intricate inquiries. Each challenge is thoroughly scrutinized to provide a comprehensive grasp of the encountered hurdles. To enhance the performance of Large Language Models (LLMs) and elevate precision, the article proposes seven potential solutions. These solutions are meticulously tailored to tackle the identified challenges and enhance the model's overall reliability. The fundamental aim of this research is to cultivate an inclusive environment conducive to evaluating the trustworthiness of ChatGPT's code generation outputs.

**Acknowledgments.** This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project-ID 490998901.

## References

1. Chen, M., et al.: Evaluating large language models trained on code. arXiv preprint [arXiv:2107.03374](https://arxiv.org/abs/2107.03374) (2021)
2. Derner, E., Batistič, K.: Beyond the safeguards: exploring the security risks of ChatGPT. arXiv preprint [arXiv:2305.08005](https://arxiv.org/abs/2305.08005) (2023)
3. Dinesh, K., Nathan, S.: Study and analysis of chat GPT and its impact on different fields of study (2023)
4. Feng, Y., Vanam, S., Cherukupally, M., Zheng, W., Qiu, M., Chen, H.: Investigating code generation performance of chat-GPT with crowdsourcing social data. In: Proceedings of the 47th IEEE Computer Software and Applications Conference, pp. 1–10 (2023)
5. Ghofrani, J., Heravi, P., Babaei, K.A., Soora-ti, M.D.: Trust challenges in reusing open source software: an interview-based initial study. In: Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume B, pp. 110–116 (2022)
6. Groppe, J., Groppe, S., Möller, R.: Variables are a curse in software vulnerability prediction. In: The 34th International Conference on Database and Expert Systems Applications (DEXA), Panang, Malaysia (2023)
7. Jansen, B.J., Jung, S.G., Salminen, J.: Employing large language models in survey research. *Nat. Lang. Process. J.* **4**, 100020 (2023)
8. Ji, Z., et al.: Survey of hallucination in natural language generation. *ACM Comput. Surv.* **55**(12), 1–38 (2023)
9. Khorashadizadeh, H., Mihindukulasooriya, N., Tiwari, S., Groppe, J., Groppe, S.: Exploring in-context learning capabilities of foundation models for generating knowledge graphs from text. arXiv preprint [arXiv:2305.08804](https://arxiv.org/abs/2305.08804) (2023)
10. Kulal, S., et al.: SPOC: search-based pseudocode to code. In: *Advances in Neural Information Processing Systems*, vol. 32 (2019)
11. Liu, C., et al.: Improving ChatGPT prompt for code generation. arXiv preprint [arXiv:2305.08360](https://arxiv.org/abs/2305.08360) (2023)

12. Magnusson, E., Marecek, J.: *Doing Interview-based Qualitative Research: A Learner's Guide*. Cambridge University Press, Cambridge (2015)
13. Mylrea, M., Robinson, N.: Ai trust framework and maturity model: improving security, ethics and trust in AI. *Cybersecur. Innov. Technol. J.* **1**(1), 1–15 (2023)
14. Parnell, K.J., et al.: Trustworthy UAV relationships: applying the schema action world taxonomy to UAVs and UAV swarm operations. *Int. J. Hum.-Comput. Interact.* **39**, 1–17 (2022)
15. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
16. Ren, S., et al.: Codebleu: a method for automatic evaluation of code synthesis. arXiv preprint [arXiv:2009.10297](https://arxiv.org/abs/2009.10297) (2020)
17. Saunders, M.N., Townsend, K.: Reporting and justifying the number of interview participants in organization and workplace research. *Br. J. Manag.* **27**(4), 836–852 (2016)
18. Sun, W., et al.: Automatic code summarization via ChatGPT: how far are we? arXiv preprint [arXiv:2305.12865](https://arxiv.org/abs/2305.12865) (2023)
19. Tao, H., Cao, Q., Chen, H., Xian, Y., Shang, S., Niu, X.: A novel software trustworthiness evaluation strategy via relationships between criteria. *Symmetry* **14**(11), 2458 (2022)
20. Tao, H., Fu, L., Chen, Y., Han, L., Wang, X.: Improved allocation and reallocation approaches for software trustworthiness based on mathematical programming. *Symmetry* **14**(3), 628 (2022)
21. Wermke, D., et al.: “Always contribute back”: a qualitative study on security challenges of the open source supply chain. In: *Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P 2023)*. IEEE (2023)