

ChatGPT for Vulnerability Detection, Classification, and Repair: How Far Are We?

Michael Fu
Monash University
Clayton, Australia
yeh.fu@monash.edu

Chakkrit (Kla) Tantithamthavorn
Monash University
Clayton, Australia
chakkrit@monash.edu

Van Nguyen
Monash University
Clayton, Australia
Van.Nguyen1@monash.edu

Trung Le
Monash University
Clayton, Australia
trunglm@monash.edu

Abstract—Large language models (LLMs) like ChatGPT (i.e., gpt-3.5-turbo and gpt-4) exhibited remarkable advancement in a range of software engineering tasks associated with source code such as code review and code generation. In this paper, we undertake a comprehensive study by instructing ChatGPT for four prevalent vulnerability tasks: function and line-level vulnerability prediction, vulnerability classification, severity estimation, and vulnerability repair. We compare ChatGPT with state-of-the-art language models designed for software vulnerability purposes. Through an empirical assessment employing extensive real-world datasets featuring over 190,000 C/C++ functions, we found that ChatGPT achieves limited performance, trailing behind other language models in vulnerability contexts by a significant margin. The experimental outcomes highlight the challenging nature of vulnerability prediction tasks, requiring domain-specific expertise. Despite ChatGPT’s substantial model scale, exceeding that of source code-pre-trained language models (e.g., CodeBERT) by a factor of 14,000, the process of fine-tuning remains imperative for ChatGPT to generalize for vulnerability prediction tasks. We publish the studied dataset, experimental prompts for ChatGPT, and experimental results at <https://github.com/awsm-research/ChatGPT4Vul>.

Index Terms—ChatGPT, Large Language Models, Cybersecurity, Software Vulnerability, Software Security

I. INTRODUCTION

Software vulnerabilities are weaknesses or flaws in software code that can be exploited by attackers to compromise the security of a system, gain unauthorized access, or cause unintended behavior. Recently, there have been advancements in employing language models for source code (e.g., CodeBERT, GraphCodeBERT, and CodeT5) to automatically achieve the following tasks: (1) pinpoint vulnerable functions and statements [9] within source code; (2) recognize vulnerability types to explain detected vulnerabilities [10]; (3) estimate the severity of vulnerabilities [10]; and (4) suggest repair patches [3], [11]. In particular, a deep learning-based software security tool named AIBugHunter was proposed in VSCode that achieves promising results for the aforementioned four vulnerability tasks using multiple fine-tuned language models for source code [10].

On the contrary, large language models (LLMs) like ChatGPT have effectively demonstrated their competence in tasks related to code, such as the simulation of system behavior from provided requirements, the formulation of API specifications, and the discernment of implicit assumptions within code [19].

Leveraging ChatGPT’s considerable scale, with 175 billion parameters for gpt-3.5-turbo [2] and 1.7 trillion parameters for gpt-4 [16], offers the potential for its application in vulnerability-related tasks. However, to the best of our knowledge, no comprehensive studies have been conducted to evaluate the entire vulnerability workflow, spanning from detecting vulnerabilities and explaining their types to estimating their severity and repair suggestions.

In this paper, we conduct a thorough analysis to assess ChatGPT’s ability for the four vulnerability prediction tasks mentioned above. Noteworthy is the fact that ChatGPT’s 1.7 trillion parameters surpass the count of parameters in source code-oriented pre-trained language models like CodeBERT and GraphCodeBERT by nearly 14,000 times. Therefore, the prevalent approach to utilizing ChatGPT involves furnishing it with appropriate prompts and task examples, rather than engaging in fine-tuning for these specific downstream tasks. It is important to note that the model parameters of ChatGPT remain proprietary by OpenAI, thereby precluding the possibility of fine-tuning its parameters for vulnerability tasks.

Thus, we compare prompting ChatGPT with other fine-tuned language models specifically designed for source code purposes. We conduct experiments to compare two versions of ChatGPT (i.e., gpt-3.5-turbo and gpt-4) with four competitive baseline approaches (i.e., AIBugHunter [10], CodeBERT [7], GraphCodeBERT [12], and VulExplainer [8]) designed for software vulnerability on four different vulnerability tasks. Through an extensive evaluation of ChatGPT on two vulnerability datasets (i.e., Big-Vul [6] and CVEFixes [1]) encompassing over 190,000 C/C++ functions, we answer the following four research questions:

(RQ1) How accurate is ChatGPT for function and line-level vulnerability predictions?

Results. ChatGPT achieves F1-measure of 10% and 29% and top-10 accuracy of 25% and 65%, which are the lowest compared with other baseline methods.

(RQ2) How accurate is ChatGPT for vulnerability types classification?

Results. ChatGPT achieves the lowest multiclass accuracy of 13% and 20%, which is 45%-52% lower than the best baseline.

(RQ3) How accurate is ChatGPT for vulnerability severity estimation?

Results. ChatGPT gave the most inaccurate severity estima-

tion with the highest mean squared error (MSE) of 5.4 and 5.85 while other baseline methods achieve MSE of 1.8 to 1.86.

(RQ4) How accurate is ChatGPT for automated vulnerability repair?

Results. ChatGPT failed to generate any correct repair patches while other baselines correctly repaired 7%-30% of vulnerable functions.

Novelty & Contributions. This paper represents one of the pioneering pilot studies that comprehensively assess ChatGPT’s (gpt-3.5-turbo and gpt-4) performance in vulnerability detection, vulnerability type identification, severity estimation, and patch recommendation. In addition, we conduct comparative analyses with other state-of-the-art language models, specifically fine-tuned for software vulnerability-related tasks.

II. RELATED WORK

Recently, researchers have been investigating the applicability of ChatGPT for software vulnerability tasks. Cheshkov *et al.* [4] investigated the base ChatGPT (gpt-3.5-turbo) performance for vulnerability prediction and classification using 120 samples across five different CWE-IDs. Zhang *et al.* [21] designed suitable prompts for ChatGPT to enhance its performance for vulnerability prediction. On the other hand, Napoli *et al.* [14] investigated ChatGPT’s performance for the smart contracts vulnerability correction task. Some previous studies have evaluated ChatGPT’s performance for the automated program repair (APR) task where the model was asked to fix general bugs [17], [18], [20]. In particular, Pearce *et al.* [17] assessed large language models’ performance for the program repair, however, the most advanced two versions of ChatGPT were not included in their experiments.

III. PROBLEM STATEMENT & PROMPT DESIGN

In this section, we introduce the problem statements of the four vulnerability tasks, i.e., (1) function and line-level software vulnerability prediction (SVP), (2) software vulnerability classification (SVC), (3) severity estimation, and (4) automated vulnerability repair (APR). After each problem statement, we illustrate how we design the prompts for ChatGPT to perform the prediction task.

A. Prompt ChatGPT for Software Vulnerability Prediction

Problem. We formulate vulnerability prediction as a binary classification task where the model predicts whether the input source code function is vulnerable. For vulnerable functions, we formulate the line-level vulnerability localization task as a ranking problem, where the model ranks vulnerable statements on the top to reduce the manual analysis workload for security analysts.

Prompt. We present example prompts for function and line-level vulnerability prediction in Fig 1. In the initial prompt, we provide ChatGPT with a task description focusing on function-level predictions, along with a clear instruction for return. In the subsequent prompt, we inform ChatGPT that the given function is vulnerable and request it to rank the top 10 most vulnerable-prone statements from the given function.

We provide a return template, anticipating that ChatGPT will generate an output consisting of a line number accompanied by its corresponding code statement as predictions.

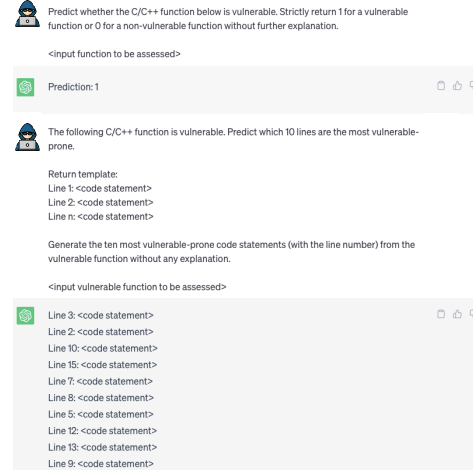


Fig. 1. An example prompt for function and line-level vulnerability prediction.

B. Prompt ChatGPT for Software Vulnerability Classification

Problem. We formulate vulnerability classification as a multi-class classification task where the model identifies a CWE-ID for an input vulnerable function. Common Weakness Enumeration Identifier (CWE-ID) is a community-developed list of common software weaknesses and vulnerabilities [5], which allows security professionals to categorize and communicate about security issues in a standardized manner.

Prompt. We present example prompts for CWE-ID classification in Fig 2. In particular, we inform ChatGPT that the input function is vulnerable and request it to identify its corresponding CWE-ID. Additionally, we limit the classification scope by providing a list of potential CWE-IDs to ensure a fair comparison with other fine-tuned models.

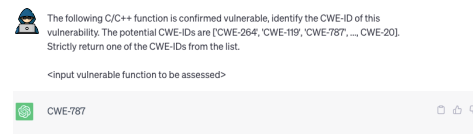


Fig. 2. An example prompt for CWE-ID classification.

C. Prompt ChatGPT for Vulnerability Severity Estimation

Problem. We formulate vulnerability severity estimation as a regression task where the model predicts a continuous value based on input vulnerable functions to estimate their severity. CVSS (Common Vulnerability Scoring System) severity score is a standardized numerical system used to assess the seriousness of security vulnerabilities in software and systems. We use CVSS version 3.1 ranging from 0 to 10.

Prompt. We present example prompts for severity estimation in Fig 3. We inform ChatGPT that the input function is vulnerable and specify the CVSS version and the output range to make it generate a severity estimation for the given function.

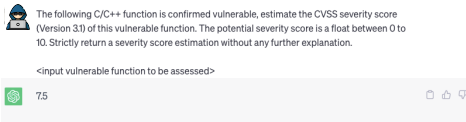


Fig. 3. An example prompt for severity estimation.

D. Prompt ChatGPT for Automated Vulnerability Repair

Problem. We formulate vulnerability repair as a sequence-to-sequence generation task where the model generates corresponding repair patches for input vulnerable functions.

Prompt. We present example prompts for vulnerability repair in Fig 4. Given that model outputs are repair patches designed by Chen *et al.* [3] instead of the complete repaired program, we provide three repair examples in each prompt to make ChatGPT comprehend our repair task. We then request ChatGPT to create repair patches for vulnerable functions using the templates provided in those examples.

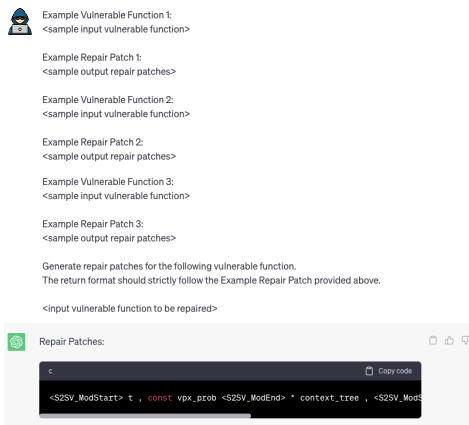


Fig. 4. An example prompt for automated vulnerability repair.

IV. EXPERIMENTAL DESIGN AND RESULTS

In this section, we introduce our experimental datasets selected for each vulnerability task followed by the parameter settings and hardware environment used to reproduce the baseline language models fine-tuned for vulnerability tasks. Finally, we present our experimental approach along with the results for each research question.

A. Experimental Datasets

We use the Big-Vul dataset constructed by Fan *et al.* [6] to evaluate vulnerability prediction (RQ1), classification (RQ2), and severity estimation (RQ3). Big-Vul has been widely adopted for software vulnerability tasks [9], [10], which comprises 188k C/C++ functions gathered from 348 Github projects, encompassing 3,754 code vulnerabilities across 91 types. Each vulnerable function is labeled with a CWE-ID and a CVSS severity score. Its data distribution mirrors real-world conditions, with a vulnerable-to-benign function ratio of 1:20. Similar to previous studies [9], [10], we split the data into 80% for training, 10% for validation, and 10% for testing.

For the automated vulnerability repair (RQ4), we leverage the Big-Vul and CVEFixes [1] datasets pre-processed by Chen *et al.* [3]. The dataset has been adopted to evaluate vulnerability repair approaches [3], [11], which contains 5.5k pairs of vulnerable functions and their repair patches. Similar to previous studies [3], [11], we split the data into 70% for training, 10% for validation, and 20% for testing.

B. Parameter Settings and Execution Environment

We replicate the baseline methods using the original authors' specified parameter settings, running experiments on a Linux machine equipped with an AMD Ryzen 9 5950X processor, 64 GB RAM, and an NVIDIA RTX 3090 GPU. The ChatGPT prompting was completed via paid API access provided by OpenAI [15].

C. Experimental Results

(RQ1) How accurate is ChatGPT for function and line-level vulnerability predictions?

Approach. To answer this RQ, we focus on the function and line-level vulnerability predictions and compare ChatGPT (i.e., gpt-3.5-turbo and gpt-4) with three other fine-tuned baseline language models as follows:

- 1) AIBugHunter: A recently proposed deep learning-based software security tool that utilizes fine-tuned language models [9], [11] to perform vulnerability prediction, classification, severity estimation, and repair [10].
- 2) CodeBERT: A language model originally pre-trained for tasks related to source code, CodeBERT underwent pre-training using the Codesearchnet dataset [13], encompassing various programming languages. CodeBERT has demonstrated its capability to effectively perform tasks associated with source code [7].
- 3) GraphCodeBERT: A language model that was also pre-trained on the Codesearchnet dataset to perform source code-related tasks. Notably, when forming the input for the model, GraphCodeBERT considers the data flow graph in addition to source code tokens [12].

Similar to the previous study [9], we report F1-measure, precision, and recall to evaluate function-level performance. For line-level performance, we report top-10 accuracy that measures the percentage of vulnerable functions where at least one actual vulnerable line appears in the model's top-10 ranking. This metric has been previously used to evaluate the prediction of line-level vulnerability prediction [9].

Result. Fig 5 presents the experimental results of function and line-level vulnerability prediction. **ChatGPT failed to accurately predict at the function level with an F1-measure of 10% and top-10 accuracy of 25% at the statement level.** The leading-edge gpt-4 with 1.7 trillion parameters achieves an F1-measure of 29% along with a top-10 accuracy of 65%. In contrast, the fine-tuned AIBugHunter achieves an F1-measure of 94% along with a top-10 accuracy of 99% with only 120 million parameters. Despite gpt-4's extensive model size and pre-training data, it faced challenges in generalizing the vulnerability prediction task without undergoing fine-tuning.

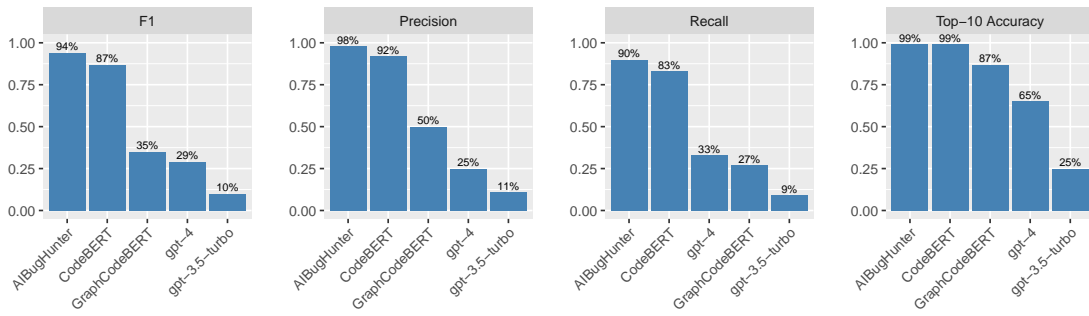


Fig. 5. (RQ1) The experimental results of function-level and line-level vulnerability prediction. (↗) For all metrics, higher = better.

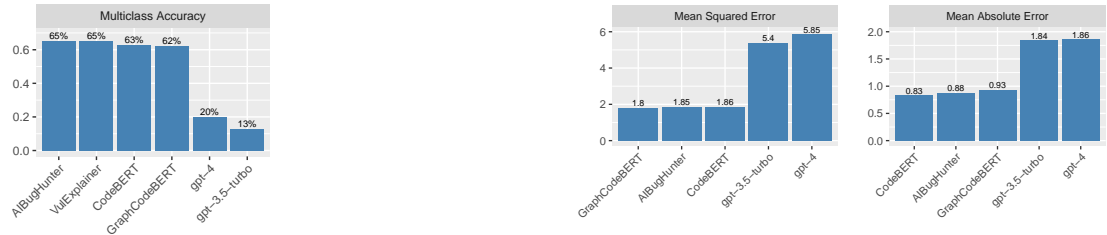


Fig. 6. (RQ2) The experimental results of vulnerability type (i.e., CWE-ID) classification. (↗) Higher Multiclass Accuracy = better.

Fig. 7. (RQ3) The experimental results of vulnerability severity estimation. (↘) Lower MSE, MAE = better.

These results highlight that the vulnerability prediction task requires models to learn domain-specific knowledge (e.g., vulnerability patterns) and fine-tuning is still required for large language models despite their significant model size.

(RQ2) How accurate is ChatGPT for vulnerability types classification?

Approach. To answer this RQ, we focus on the vulnerability classification task where we aim to identify CWE-IDs for vulnerable functions. We compare ChatGPT (i.e., gpt-3.5-turbo and gpt-4) with three fine-tuned baseline language models introduced in RQ1. Additionally, we include VulExpainer [8] which leverages language models with a distillation framework to mitigate the data imbalances in the CWE-ID classification task. Similar to previous studies [8], [10], we use the multiclass accuracy measure to evaluate the performance of each method.

Result. Fig 6 presents the experimental results of CWE-ID classification. **The accuracy of ChatGPT in correctly identifying CWE-IDs for vulnerable functions is limited, standing at a mere 13%.** The gpt-3.5-turbo and gpt-4 achieve 13%-20% accuracy while the fine-tuned language model baselines achieve 62%-65%. These findings suggest that the accurate identification of CWE-ID for a vulnerable function requires the model to learn to map specific patterns (e.g., buffer overflow) in vulnerable functions to a CWE-ID. However, ChatGPT has not adequately acquired such knowledge during the pre-training phase of ChatGPT so a fine-tuning stage is still required to boost its performance.

(RQ3) How accurate is ChatGPT for vulnerability severity estimation?

Approach. To answer this RQ, we focus on predicting the CVSS score of vulnerable functions. We compare ChatGPT (gpt-3.5-turbo and gpt-4) with the three baselines introduced in RQ1, where CodeBERT has been shown to be effective for severity estimation [10]. Similar to the previous study [10], we use Mean Squared Error (MSE) and Mean Absolute Error (MAE) to assess the performance of each method.

Result. Fig 7 presents the experimental results of the severity score estimation. **ChatGPT failed to accurately estimate the CVSS severity score, resulting in an MSE of 5.4 and an MAE of 1.84.** The gpt-3.5-turbo and gpt-4 have MSE of 5.4 and 5.85 while the fine-tuned language model baselines achieve 1.8-1.86. Similar to vulnerability prediction and classification tasks, accurately estimating severity scores also demands software security expertise that ChatGPT has not acquired during its extensive pre-training phase, hindering its ability to provide accurate predictions in this context.

(RQ4) How accurate is ChatGPT for automated vulnerability repair?

Approach. To answer this RQ, we focus on generating vulnerability repair patches for vulnerable functions. We compare ChatGPT (i.e., gpt-3.5-turbo and gpt-4) with the three baseline methods introduced in RQ1. Notably, AIBugHunter leverages the VulRepair [11] model for repair patches generation, which achieves state-of-the-art results in the vulnerability repair problem. Similar to previous studies [3], [11], we use the percentage of perfect prediction (%PP) measure to assess the performance of each method. Only if the generated repair patches

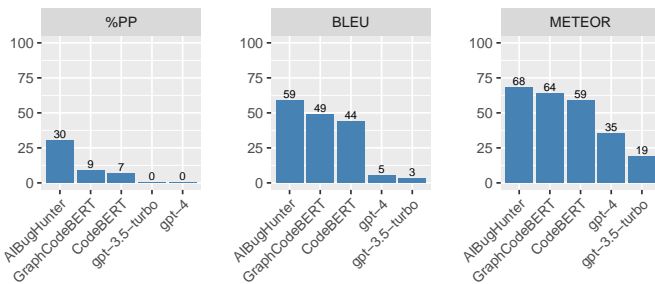


Fig. 8. (RQ4) The experimental results of automated vulnerability repair. (↗) Higher %PP, BLEU, METEOR = better.

are identical to the ground-truth patches, we count it as a correct prediction. The %PP is computed as $\frac{\text{total correct predictions}}{\text{total testing samples}}$. We use greedy decoding to return one repair candidate for fine-tuned language models and ensure a fair comparison with ChatGPT. Furthermore, we incorporate BLEU and METEOR scores to evaluate the degree of similarity between the patches produced by the model and the actual patches.

Result. Fig 8 presents the experimental results of the vulnerability repair. **ChatGPT failed to generate correct repair patches for all of the vulnerable functions in our testing data.** In contrast, the fine-tuned language model baselines can correctly repair 7%-30% of the testing function. The BLEU and METEOR scores further demonstrate that repair patches generated using baseline methods exhibit greater proximity to the true patches compared to those generated through ChatGPT methods. These results indicate that vulnerability repair is a more challenging task compared with other vulnerability prediction tasks, where ChatGPT struggle to generate correct repairs for vulnerable functions. Thus, a fine-tuning step using domain-specific data is crucial for ChatGPT to generalize its ability for the vulnerability repair task.

V. CONCLUSION

In this paper, we empirically evaluate the performance of prompting two versions of ChatGPT (gpt-3.5-turbo and gpt-4) for four common vulnerability tasks: locating vulnerabilities, identifying vulnerability types, estimating severity scores, and suggesting repair patches. We compare the performance of ChatGPT with other pre-trained language models that have significantly smaller model sizes than ChatGPT but have been fine-tuned to perform software vulnerability prediction tasks. Through an assessment encompassing over 190,000 real-world C/C++ functions, ChatGPT yielded the least favorable outcomes across all vulnerability-related tasks, notably struggling to generate accurate patches for the vulnerability repair task. These findings highlight the imperative of possessing security expertise in addressing software vulnerability prediction tasks, a facet not assimilated by ChatGPT during its extensive pre-training phase. Thus, an additional round of fine-tuning stands as a pivotal requirement for ChatGPT to effectively generalize and undertake software vulnerability tasks.

REFERENCES

- [1] G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2021, pp. 30–39.
- [2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [3] Z. Chen, S. Kommrusch, and M. Monperrus, "Neural transfer learning for repairing security vulnerabilities in c code," *IEEE Transactions on Software Engineering (TSE)*, 2022. [Online]. Available: <https://arxiv.org/pdf/2104.08308>
- [4] A. Cheshkov, P. Zadorozhny, and R. Leviceh, "Evaluation of chatgpt model for vulnerability detection," *arXiv preprint arXiv:2304.07232*, 2023.
- [5] M. Corporation, "Common weakness enumeration (cwe)," <https://cwe.mitre.org/index.html>, 2006.
- [6] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A c/c++ code vulnerability dataset with code changes and cve summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, 2020, pp. 508–512.
- [7] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1536–1547.
- [8] M. Fu, V. Nguyen, C. K. Tantithamthavorn, T. Le, and D. Phung, "Vulexplainer: A transformer-based hierarchical distillation for explaining vulnerability types," *IEEE Transactions on Software Engineering*, pp. 1–17, 2023.
- [9] M. Fu and C. Tantithamthavorn, "Linevul: A transformer-based line-level vulnerability prediction," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. IEEE, 2022.
- [10] M. Fu, C. Tantithamthavorn, T. Le, Y. Kume, V. Nguyen, D. Phung, and J. Grundy, "Aibughunter: A practical tool for predicting, classifying and repairing software vulnerabilities," *arXiv preprint arXiv:2305.16615*, 2023.
- [11] M. Fu, C. Tantithamthavorn, T. Le, V. Nguyen, and D. Phung, "Vulrepair: a t5-based automated software vulnerability repair," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2022, pp. 935–947.
- [12] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, L. Shujie, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," in *International Conference on Learning Representations*, 2021.
- [13] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.
- [14] E. A. Napoli and V. Gatteschi, "Evaluating chatgpt for smart contracts vulnerability correction," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2023, pp. 1828–1833.
- [15] OpenAI, "Chatgpt," <https://openai.com/blog/chatgpt>, 2022.
- [16] —, "Gpt-4 technical report," 2023.
- [17] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2339–2356.
- [18] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," *arXiv preprint arXiv:2301.08653*, 2023.
- [19] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," *arXiv preprint arXiv:2303.07839*, 2023.
- [20] C. S. Xia and L. Zhang, "Keep the conversation going: Fixing 162 out of 337 bugs for \$0.42 each using chatgpt," *arXiv preprint arXiv:2304.00385*, 2023.
- [21] C. Zhang, H. Liu, J. Zeng, K. Yang, Y. Li, and H. Li, "Prompt-enhanced software vulnerability detection using chatgpt," *arXiv preprint arXiv:2308.12697*, 2023.