

Singapore Management University

## Institutional Knowledge at Singapore Management University

---

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

---

4-2024

### Large language model for vulnerability detection: Emerging results and future directions

Xin ZHOU

Singapore Management University, xinzhou.2020@phdcs.smu.edu.sg

Ting ZHANG

Singapore Management University, tingzhang@smu.edu.sg

David LO

Singapore Management University, davidlo@smu.edu.sg

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)



Part of the [Programming Languages and Compilers Commons](#), and the [Software Engineering Commons](#)

---

#### Citation

ZHOU, Xin; ZHANG, Ting; and LO, David. Large language model for vulnerability detection: Emerging results and future directions. (2024). *ICSE-NIER'24: Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, Lisbon, Portugal, April 14-20*. 47-51.

Available at: [https://ink.library.smu.edu.sg/sis\\_research/9245](https://ink.library.smu.edu.sg/sis_research/9245)

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [cherylds@smu.edu.sg](mailto:cherylds@smu.edu.sg).



# Large Language Model for Vulnerability Detection: Emerging Results and Future Directions

Xin Zhou, Ting Zhang, and David Lo

School of Computing and Information Systems, Singapore Management University, Singapore  
{xinzhou.2020,tingzhang.2019}@phdcs.smu.edu.sg,davidlo@smu.edu.sg

## ABSTRACT

Previous learning-based vulnerability detection methods relied on either medium-sized pre-trained models or smaller neural networks from scratch. Recent advancements in Large Pre-Trained Language Models (LLMs) have showcased remarkable few-shot learning capabilities in various tasks. However, the effectiveness of LLMs in detecting software vulnerabilities is largely unexplored. This paper aims to bridge this gap by exploring how LLMs perform with various prompts, particularly focusing on two state-of-the-art LLMs: GPT-3.5 and GPT-4. Our experimental results showed that GPT-3.5 achieves competitive performance with the prior state-of-the-art vulnerability detection approach and GPT-4 consistently outperformed the state-of-the-art.

## 1 INTRODUCTION

Software vulnerabilities are the prevalent issues in software systems, posing various risks such as the compromise of sensitive information [1] and system failures [2]. To address this challenge, researchers have proposed machine learning (ML) and deep learning (DL) approaches for identifying vulnerabilities in source code [3–6]. While previous ML/DL-based vulnerability detection approaches have demonstrated promising results, they have primarily relied on either medium-size pre-trained models such as CodeBERT [4, 7] or training smaller neural networks (such as Graph Neural Networks [5]) from scratch.

Recent developments in Large Pre-Trained Language Models (LLMs) have demonstrated impressive few-shot learning capabilities across diverse tasks [8–12]. However, the performance of LLMs on security-oriented tasks, particularly vulnerability detection, remains largely unexplored. Moreover, LLMs are gradually starting to be used in software engineering (SE), as seen in automated program repair [8]. However, these studies predominantly focus on using LLMs for generation-based tasks. It remains unclear whether LLMs can be effectively utilized in classification tasks and outperform the medium-size pre-trained models such as CodeBERT, specifically in the vulnerability detection task.

To fill in the research gaps, this paper investigates the effectiveness of LLMs in identifying vulnerable codes, i.e., a critical classification task within the domain of security. Furthermore, the efficacy of LLMs heavily relies on the quality of prompts (task descriptions and other relevant information) provided to the model. Thus, we

explore and design diverse prompts to effectively apply LLMs for vulnerability detection. We specifically studied two prominent state-of-the-art Large Language Models (LLMs): **GPT-3.5** and **GPT-4**, both serving as the foundational models for ChatGPT. Our experimental results revealed that with appropriate prompts, GPT-3.5 achieves competitive performance with CodeBERT, and GPT-4 outperformed CodeBERT by 34.8% in terms of Accuracy. In summary, our contributions are as follows:

- We conduct experiments with diverse prompts for LLMs, encompassing task and role descriptions, project information, and examples from Common Weakness Enumeration (CWE) and the training set. We recognize LLMs as promising models for vulnerability detection.
- We pinpoint several promising future directions for leveraging LLMs in vulnerability detection, and we encourage the community to delve into these possibilities.

## 2 PROPOSED APPROACH

**ChatGPT and In-Context Learning.** ChatGPT (Plus) is built upon closed-source large-size LLMs known as the GPT-3.5 and GPT-4. Much prior research employs medium-size pre-trained models like CodeBERT and CodeT5. These models are commonly fine-tuned, updating all parameters to align with labeled training data. [13, 14]. Though very effective, fine-tuning demands large GPU resources to load and update all parameters of pre-trained models [11, 15]. As large-size LLMs (e.g., ChatGPT) have a large number of parameters, it is very challenging to fine-tune them using GPU cards widely used in academia. An alternative and widely adopted approach for large-size LLMs, as introduced in GPT-3, is *in-context learning* (ICL) [12, 16]. ICL involves freezing the parameters of LLMs and utilizing suitable prompts to impart task-specific knowledge to the models. Unlike fine-tuning, ICL requires no parameter update which significantly reduces the large GPU resource requirement. To perform the inference/testing, ICL makes predictions based on the probability of generating the next token  $t$  given the unlabeled data instance  $x$  and the prompt  $P$ . Then the output token  $t$  is mapped into the prediction categories by the verbalizer (introduced below).

**Prompt Basics.** A prompt is a textual string that has two slots: (1) an input slot  $[X]$  for the original input data  $x$  and (2) an answer slot  $[Z]$  for the predicted answer  $z$ . The verbalizer, denoted as  $V$ , is a function that maps the predicted answer  $z$  to a class  $\hat{y}$  in the target class set  $Y$ , formally  $V : Z \rightarrow Y$ . For instance, a straightforward prompt and verbalizer are shown as follows:  $f_{prompt}(x) =$  “Code is  $[X]$ . It is  $[Z]$ .” and  $V = \begin{cases} +, & \text{if } Z = \text{vulnerable}, \\ -, & \text{if } Z = \text{non-vulnerable}, \end{cases}$

where  $V$  is the defined verbalizer where the token “vulnerable” is mapped into the positive class. ChatGPT may generate responses that differ from our predefined label words. To simplify the process,



This work licensed under Creative Commons Attribution International 4.0 License.

ICSE-NIER'24, April 14–20, 2024, Lisbon, Portugal  
© 2024 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-0500-7/24/04.  
<https://doi.org/10.1145/3639476.3639762>

**Table 1: Our prompt designs.**

No.	Prompt Type	Prompt Template	verbalizer
P	Task Description	Now you need to identify whether a method contains a vulnerability or not. If has any potential vulnerability, output: 'this code is vulnerable'. Otherwise, output: 'this code is non-vulnerable'. The code is [X]. Let's start: [Z]	+: this code is vulnerable -: this code is non-vulnerable
A1	Role Description	You are an experienced developer who knows the security vulnerability very well	+: this code is vulnerable -: this code is non-vulnerable
A2	Project Information	The code is from [the name of project]. The filename is [the name of file].	+: this code is vulnerable -: this code is non-vulnerable
A3	Dangerous CWE Types Examples	Here are examples of the most dangerous CWE types. Example1: int returnChunkSize ... Label1: this code is vulnerable. Example2: static ... ...	+: this code is vulnerable -: this code is non-vulnerable
A4	Randomly Sampled Code	Here are sampled examples from the training data. Example1: int ... Label1: this code is vulnerable. Example2: static ... Label2: this code is non-vulnerable. ...	+: this code is vulnerable -: this code is non-vulnerable
A5	Retrieved Similar Code	Here are the most similar codes from the training data. Example1: int ... Label1: this code is non-vulnerable. Example2: static ... Label2: this code is vulnerable. ...	+: this code is vulnerable -: this code is non-vulnerable

we manually check the predicted classes of ChatGPT’s generated answers when they diverge from our specified label words. For example, we map the answer "it is vulnerable because ..." into the "vulnerable" class.

**Prompt Designs.** Table 1 shows our designed prompts (the base prompt + several augmentations). OpenAI allows users to guide ChatGPT through two types of messages/prompts: 1) the system message, influencing ChatGPT’s overall behaviors such as adjusting the personality of ChatGPT, and 2) the user message, containing requests for ChatGPT to address and respond to. We use an empty system message and user messages that are listed as prompts in Table 1. Initially, we designed a straightforward prompt (*P*): "Now you need to identify whether a method contains a vulnerability or not." as the base prompt. This base prompt only briefly describes the task we want LLMs to do. To provide LLMs with more valuable task-specific information, we propose diverse **augmentations** (*A\**) to the base prompt, including the following:

*Role Description (A1):* We explicitly defined the role of LLMs in this task: "You are an experienced developer who knows the security vulnerability very well". This strategy aims to remind LLMs to change their working mode to a security-related one.

*Project Information (A2):* Recently, Li et al. [17] propose the state-of-the-art LLM for code namely StarCoder. Li et al. found that adding the filename in the prompts can substantially improve the effectiveness of StarCoder. We followed them to provide LLMs with the project names and filenames associated with the target code.

*External Source Knowledge (A3):* The CWE system offers a wealth of information about software vulnerabilities such as code examples of typical vulnerable code. Leveraging such resources could possibly enhance the prompt generation process for vulnerability detection tasks. In this study, we collected the vulnerable code examples that represent the top 25 most dangerous Common Weakness Enumeration (CWE) types identified in the year 2022 [18]. These examples showcase the characteristics and patterns of vulnerabilities, equipping LLMs with valuable insights. This allows us to extend the

model’s knowledge beyond the limitations of the training data by leveraging external sources, specifically the CWE system.

*Knowledge in the Training Set (A4):* The training data encompasses valuable task-specific knowledge pertinent to a given task. However, we can only accommodate a limited number of input-output samples because ChatGPT has a maximum token limit of 4,096. In this strategy, we randomly select *K* samples from the training data to leverage the knowledge embedded within the training dataset. Vulnerable/non-vulnerable samples are both used in this strategy.

*Selective Knowledge in the Training Set (A5):* In contrast to the aforementioned strategy, we adopted a different approach by retrieving the top *K* most similar methods from the training data, instead of randomly sampling. These retrieved methods served as examples to furnish LLMs with pertinent knowledge, aiding their decision-making process when evaluating the test data. To perform the retrieval process, we employed CodeBERT [7] to transform the code snippets into semantic vectors. Subsequently, we quantified the similarity between two code snippets by calculating the cosine similarity of their respective semantic vectors. For a given test code, we retrieved the top *K* similar methods along with their corresponding vulnerability labels from the training data.

### 3 PRELIMINARY EVALUATION

In this work, we aim to answer a single research question: **How effective is ChatGPT with different prompt designs in vulnerability detection compared to baselines?**

**Dataset and Model.** We use the vulnerability-fixing commit dataset recently collected by Pan et al. [19]. To get the vulnerable functions from vulnerability-fixing commits, we followed Fan et al. [20] to first collect software versions prior to a vulnerability-fixing commit, and then labeled functions with lines changed in a patch as vulnerable. All remaining functions in a file touched by a commit were regarded as non-vulnerable. As the security patches dataset covers a large number of software repositories implemented in diverse programming languages, it is challenging to write all the corresponding parsers (used to split functions from files) for all languages. Thus, in this preliminary evaluation, we only focus on software repositories implemented in C/C++. To build our test set, we first randomly sampled 20 open-source software repositories implemented in C/C++ from the original test set split by Pan et al. [19] and used their vulnerability fixes to get the vulnerable functions (positive samples) for our test set. Due to the considerable cost [21] associated with querying ChatGPT on a large test set, we limited our sampling to 20 repositories. Despite the restricted sample size, our preliminary experiment was designed to showcase the potential of ChatGPT. For our training/validation sets, we used all the vulnerability fixes of the C/C++ repositories in the original training/validation sets split by Pan et al. and extracted vulnerable functions (positive samples). To obtain negative samples, i.e., non-vulnerable functions, for the test/training/validation sets, we employed a random sampling technique. For each vulnerable function, we selected one function at random from non-vulnerable functions that were extracted from the same file as the vulnerable function. Different from the vulnerable function, those non-vulnerable functions had not been modified by the vulnerability-fixing commit. Finally, our dataset has 7,683/853/368 methods in the training/validation/test sets.

**Table 2: Results of ChatGPT with diverse prompts and the fine-tuned CodeBERT**

Model	Prompt	Prompt/Model Description	Param.	Accuracy	Precision	Recall	F1 score	F0.5 score
ChatGPT (GPT-3.5)	P	provide the task description to LLM	-	50.0	Nan	0.0	Nan	Nan
	P+A1	provide the role description to LLM	-	50.0	Nan	0.0	Nan	Nan
	P+A2	provide the project name	-	50.0	Nan	0.0	Nan	Nan
	P+A3	provide vulnerable code examples from 25 most dangerous CWE Types in 2022	-	59.1	72.2	29.5	41.9	56.0
	P+A4	randomly sample K input-output pairs	K=1	51.8	75.2	6.0	10.7	20.5
			K=3	58.8	65.8	41.2	50.2	58.4
			K=5	61.4	<b>80.0</b>	30.3	43.9	60.1
	P+A5	retrieve top K most similar code	K=1	55.4	67.2	21.2	32.3	46.9
			K=3	56.7	60.1	39.9	48.0	54.6
			K=5	59.8	63.2	47.2	54.0	59.2
P+A4+A5	combine P4 and P5 together (both top 3)	K=3	<b>62.7</b>	76.3	36.8	49.7	<b>62.8</b>	
CodeBERT	-	full-parameter fine-tuned	-	60.3	62.3	<b>53.3</b>	<b>57.3</b>	60.1

**Table 3: Results of GPT-4 and baselines on the first half of the test set**

Model	Prompt	Prompt/Model Description	Param.	Accuracy	Precision	Recall	F1 score	F0.5 score
ChatGPT (GPT-3.5)	P+A4+A5	combine P4 and P5 together	K=3	63.6	<b>77.8</b>	38.0	51.1	64.3
ChatGPT (GPT-4)	P	provide the task description to LLM	-	60.3	67.3	40.2	50.3	59.3
	P+A3	code examples from CWE Types	-	<b>75.5</b>	73.7	<b>79.3</b>	<b>76.4</b>	<b>74.8</b>
	P+A5	retrieve top K most similar code	K=5	61.4	63.6	53.3	58.0	61.3
	P+A4+A5	combine A4 and A5 together	K=3	59.2	60.2	54.3	57.1	59.0
CodeBERT	-	fine-tuned, tested on a half test set	-	56.0	57.3	46.7	51.5	54.8

For studied models, we primarily focused our investigation on ChatGPT (GPT-3.5) with the model name gpt-3.5-turbo while also doing limited experiments on ChatGPT (GPT-4). For the baseline model, we opted for one of the state-of-the-art approaches (i.e., CodeBERT) according to a recent comprehensive empirical study [22].

**Evaluation.** To measure the model’s effectiveness, we adopted widely used evaluation metrics, i.e., Accuracy, Precision, Recall, F1, and F0.5. We incorporated the F0.5 metric, which assigns greater importance to precision than to recall. This choice is motivated by the developers’ aversion to false positives, as a low success rate may diminish their patience and confidence in the system [23]. As the ICL method exhibits some instability, in this preliminary work, we repeated experiments twice and reported the average results.

**Results.** The performance of GPT-3.5 in vulnerability detection was evaluated by integrating different prompts, and the results are summarized in Table 2. Experimental results revealed that the base prompt yielded unsatisfactory outcomes, with GPT-3.5 predicting every target code as non-vulnerable, resulting in an accuracy of 50% and a recall of 0%. The inclusion of role descriptions and project information did not contribute to better performance. However, incorporating examples from external source knowledge, specifically the 25 most dangerous CWE types, led to substantial performance improvements (18.2% in Accuracy). Furthermore, the utilization of random sampling codes from the training data and retrieving similar codes also resulted in significantly better performance (up to 22.8% and 19.6% in Accuracy) compared to the base prompt. Among all the prompt combinations studied, the P+A5 combination achieved the highest F1 score (54.0%) and Recall (47.2%), and the P+A4+A5 combination achieved the best F0.5 (62.8%) score and Accuracy (62.7%). The P+A4+A5 combination outperforms the base prompt P by 25.4% in Accuracy. When comparing GPT-3.5 to the state-of-the-art approach, GPT-3.5 (P+A4+A5) outperformed CodeBERT by 4.0%, 22.5%, and 4.5% in terms of Accuracy, Precision, and

F0.5, respectively. However, GPT-3.5 underperformed CodeBERT by 44.8% and 15.3% in Recall and F1. These experimental results highlight the distinct strengths of CodeBERT and GPT-3.5. GPT-3.5 demonstrates significantly higher Precision scores, indicating its proficiency in minimizing false positives. On the other hand, CodeBERT showcases a much higher Recall score, signifying its capability to identify a greater number of vulnerabilities. **Overall, GPT-3.5 demonstrates competitive performance when compared to the fine-tuned CodeBERT.**

The performance of GPT-4 is presented in Table 3. Notably, GPT-4 is accompanied by a considerably higher cost. Due to the high costs, in this preliminary evaluation, we only evaluate GPT-4 in the first half of the test set. To assess its performance, we employed four different prompts: the base prompt (P), the external source knowledge prompt P+A3, the prompt P+A5 which obtained the second best F1 for GPT-3.5, and the prompt P+A4+A5 which obtained the best F1 for GPT-3.5. As illustrated in Table 3, **GPT-4 with the prompt P+A3 significantly outperformed the fine-tuned CodeBERT by 34.8% in terms of accuracy.**

## 4 THREATS TO VALIDITY

One concern arises due to the possibility of data leakage in ChatGPT. The dataset used in our evaluation may overlap with the data used for training ChatGPT. However, because ChatGPT is a closed-source model, we lack the means to validate whether such an overlap exists. Another threat to validity is the equal data ratio between vulnerable and non-vulnerable functions in the test set. We create a balanced test set to alleviate costs linked to ChatGPT usage, given that the expenses of these experiments rise in proportion to the number of test samples, particularly with a large number of non-vulnerable functions. However, the equal data ratio does not realistically represent the actual problem of vulnerability prediction where the vulnerable code is the minority in software systems. Besides, this data ratio will lead to inflated metrics in this study. We recognize this issue as a limitation of the preliminary



study, and in our future work, we aim to develop a new test set that accurately reflects the real-world data ratio between vulnerable and non-vulnerable functions. Lastly, we followed Fan et al. [20] to label functions with lines changed in a vulnerability-fixing commit as vulnerable. This labeling heuristic may overestimate the actual number of vulnerable functions due to the commit tangling, where some changed functions in the vulnerability-fixing commit may not be directly relevant to addressing the vulnerability. We recognize the challenge of accurately identifying real vulnerability-related functions in the vulnerability-fixing commit. In our future work, we aim to employ existing techniques (e.g., [24]) to untangle the vulnerability-fixing commits.

## 5 RELATED WORK

We are unable to find vulnerability detection work using GPT-3.5 and GPT-4 in the academic literature. However, there are several related studies in the gray (non-peer-reviewed) literature. For example, BurpGPT [25] integrates ChatGPT with the Burp Suite [26] to detect vulnerabilities in web applications. In contrast, the software studied in our study is not confined to a specific domain. Also, vuln\_GPT, an LLM designed to discover and address software vulnerabilities, was introduced recently [27]. Different from [27], our study focuses on improving prompts for vulnerability detection. A parallel work [28] also explores the use of ChatGPT in vulnerability detection. They mainly enhance prompts through structural and sequential auxiliary information. Three distinctions of our work from [28] are: 1) We also studied GPT-4 while they did not; 2) We integrate knowledge from the CWE system and similar samples from the training set to enhance prompts, a dimension not considered in [28]; 3) We identified promising future study directions which are not thoroughly discussed in [28].

## 6 FUTURE WORK

There are plenty of exciting paths to explore in future research. Here, we're just highlighting a few of these potential directions:

**Local and Specialized LLMs-based Vulnerability Detection.** This study focuses on ChatGPT. However, ChatGPT requires data to be sent to third-party services. This may restrict the utilization of ChatGPT-related vulnerability detection tools among specific organizations, such as major tech corporations or governments. These organizations, e.g., [29–31], regard their source code as proprietary, sensitive, or classified material, and as a result, they are unable to transmit or share it with third-party services. Additionally, ChatGPT models are not specialized for vulnerability detection and thus may not take full advantage of the rich open-source vulnerability data.

To address the above-mentioned limitations, in the future, we aim to propose a local and specialized LLM solution for vulnerability detection. The solution will build upon general-purpose and open-source code LLMs, e.g., Llama [32], which will be tuned for vulnerability detection with the relevant vulnerability corpus. The tuned LLM can alleviate the concerns of organizations that prioritize data security and privacy while also making use of the abundant open-source vulnerability data.

**Precision and Robustness Boost in Vulnerability Detection.** A vulnerability detection solution with a high precision is usually preferred. Additionally, the solution needs to remain robust against

data perturbations or adversarial attacks. With higher precision, developers will have greater confidence in the reliability of detections and consequently perform the requisite actions to address the detected vulnerabilities. With higher robustness, a more secure and stable vulnerability detection model can be produced, and be more immune to adversarial attacks.

In the future, we plan to boost LLMs' precision and robustness in this task: 1) To improve precision, we plan to employ ensemble learning, a promising technique to improve the precision (and possibly recall) by identifying the common high-confidence predictions among different models. 2) To improve robustness, we plan to extend an existing work [33] that employs a single adversarial transformation (renaming of variables) to enhance model robustness. We will delve into various other types of adversarial transformations and assess their effectiveness in enhancing the robustness of LLMs.

**Enhancing Effectiveness in Long-Tailed Distribution.** In this study, we formulate the task as a binary classification (vulnerable or non-vulnerable). Moving one step further, developers may require the tool to indicate the specific vulnerability type (e.g., CWE types) associated with the detected vulnerable code. This additional information is crucial for a better understanding and resolution of the vulnerability. However, a recent study [34] revealed that vulnerability data exhibit a long-tailed distribution in terms of CWE types: a small number of CWE types have a substantial number of samples, while numerous CWE types have very few samples. The study also pointed out that LLMs struggle to effectively handle vulnerabilities in these less common types. This long-tailed distribution could pose a challenge for LLMs-based vulnerability detection solutions.

In the future, we plan to 1) explore whether LLMs, specifically ChatGPT, can effectively detect these infrequent vulnerabilities or not and 2) propose a solution (e.g., generating more samples for the less common types via data augmentation) to address the impact of the long-tailed distribution of vulnerability data.

**Trust and Synergy with Developers.** AI-powered solutions for vulnerability detection, including this work, have limited interaction with developers. They may face challenges in establishing trust and synergy with developers during practical use. To overcome this, future works should investigate more effective strategies to foster trust and collaboration between developers and AI-powered solutions [35]. By nurturing trust and synergy, AI-powered solutions may evolve into smart workmates to better assist developers.

## 7 CONCLUSION

In this study, we explored the efficacy and potential of LLMs (i.e., ChatGPT) in vulnerability detection. We proposed some insightful prompt enhancements such as incorporating the external knowledge and choosing valuable samples from the training set. We also identified many promising directions for future study. We made our replication package<sup>1</sup> publicly available for future studies.

**Acknowledgement.** This research / project is supported by the National Research Foundation, under its Investigatorship Grant (NRF-NRFI08-2022-0002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

<sup>1</sup><https://github.com/soarsmu/ChatGPT-VulDetection>

## REFERENCES

- [1] Microsoft Exchange Flaw: Attacks Surge After Code Published. <https://www.bankinfosecurity.com/ms-exchange-flaw-causes-spike-introdownloader-gen-trojans-a-16236>, 2022.
- [2] Dean Turner, Marc Fossi, Eric Johnson, Trevor Mack, Joseph Blackbird, Stephen Entwisle, Mo King Low, David McKinney, and Candid Wueest. Symantec global internet security threat report—trends for july–december 07. *Symantec Enterprise Security*, 13:1–36, 2008.
- [3] Hazim Hanif and Sergio Maffei. Vulberta: Simplified source code pre-training for vulnerability detection. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.
- [4] Michael Fu and Chakkrit Tantithamthavorn. Linevul: a transformer-based line-level vulnerability prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 608–620, 2022.
- [5] Van-Anh Nguyen, Dai Quoc Nguyen, Van Nguyen, Trung Le, Quan Hung Tran, and Dinh Phung. Regvd: Revisiting graph neural networks for vulnerability detection. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pages 178–182, 2022.
- [6] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32, 2019.
- [7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics, 2020.
- [8] Chunqiu Steven Xia and Lingming Zhang. Keep the conversation going: Fixing 162 out of 337 bugs for \$0.42 each using chatgpt. In *arXiv preprint arXiv:2304.00385*, 2023.
- [9] Ting Zhang, Ivana Claire Irsan, Ferdian Thung, and David Lo. Cupid: Leveraging chatgpt for more accurate duplicate bug report detection. *arXiv preprint arXiv:2308.10022*, 2023.
- [10] Ting Zhang, Ivana Claire Irsan, Ferdian Thung, and David Lo. Revisiting sentiment analysis for software engineering in the era of large language models. *arXiv preprint arXiv:2310.11113*, 2023.
- [11] Martin Weyssow, Xin Zhou, Kisub Kim, David Lo, and Houari Sahraoui. Exploring parameter-efficient fine-tuning techniques for code generation with large language models. *arXiv preprint arXiv:2308.10462*, 2023.
- [12] Xin Zhou, Bowen Xu, Kisub Kim, DongGyun Han, Thanh Le-Cong, Junda He, Bach Le, and David Lo. Patchzero: Zero-shot automatic patch correctness assessment. *arXiv preprint arXiv:2303.00202*, 2023.
- [13] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. Sentiment analysis for software engineering: How far can pre-trained transformer models go? In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 70–80. IEEE, 2020.
- [14] Xin Zhou, DongGyun Han, and David Lo. Assessing generalizability of codebert. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 425–436. IEEE, 2021.
- [15] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685, 2022.
- [16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [17] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.
- [18] [https://cwe.mitre.org/top25/archive/2022/2022\\_cwe\\_top25.html](https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html), 2022.
- [19] Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. Fine-grained commit-level vulnerability type prediction by cwe tree structure. In *45th International Conference on Software Engineering, ICSE 2023*, 2023.
- [20] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. Ac/c++ code vulnerability dataset with code changes and cve summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 508–512, 2020.
- [21] <https://openai.com/pricing>, 2023.
- [22] Benjamin Steenhoeck, Md Mahbubur Rahman, Richard Jiles, and Wei Le. An empirical study of deep learning models for vulnerability detection. *45th International Conference on Software Engineering, ICSE, 2023*.
- [23] Pavneet Singh Kochhar, Xin Xia, David Lo, and Shanping Li. Practitioners’ expectations on automated fault localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 165–176, 2016.
- [24] Yi Li, Shaohua Wang, and Tien N Nguyen. Utango: untangling commits with context-aware, graph-based, code change clustering learning model. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 221–232, 2022.
- [25] Alexandre Teyar. Burpgpt - chatgpt powered automated vulnerability detection tool. <https://burpgpt.app/#faq>, 2023.
- [26] PortSwigger. Burp suite - application security testing software. <https://portswigger.net/burp>.
- [27] Vicarius. vuln\_gpt debuts as ai-powered approach to find and remediate software vulnerabilities. [https://venturebeat.com/ai/got-vulns-vuln\\_gpt-debuts-as-ai-powered-approach-to-find-and-remediate-software-vulnerabilities/](https://venturebeat.com/ai/got-vulns-vuln_gpt-debuts-as-ai-powered-approach-to-find-and-remediate-software-vulnerabilities/), 2023.
- [28] Chenyuan Zhang, Hao Liu, Jiutian Zeng, Kejing Yang, Yuhong Li, and Hui Li. Prompt-enhanced software vulnerability detection using chatgpt. *arXiv preprint arXiv:2308.12697*, 2023.
- [29] Jon Harper. Pentagon testing generative AI in ‘global information dominance’ experiments. <https://defensescoop.com/2023/07/14/pentagon-testing-generative-ai-in-global-information-dominance-experiments/>, 2023.
- [30] Kyle Chua. Samsung Bans Use of Generative AI Tools on Company-Owned Devices Over Security Concerns. <https://www.tech360.tv/samsung-bans-use-generative-ai-tools>, 2023.
- [31] Kyle Chua. Apple Bans Internal Use of ChatGPT GitHub Copilot Over Fear of Leaks. <https://www.tech360.tv/apple-bans-internal-use-chatgpt-github-copilot-over-fears-of-leaks>, 2023.
- [32] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [33] Zhou Yang, Jieke Shi, Junda He, and David Lo. Natural attack for pre-trained models of code. In *Proceedings of the 44th International Conference on Software Engineering*, 2022.
- [34] Xin Zhou, Kisub Kim, Bowen Xu, Jiakun Liu, DongGyun Han, and David Lo. The devil is in the tails: How long-tailed code distributions impact large language models. *arXiv preprint arXiv:2309.03567*, 2023.
- [35] David Lo. Trustworthy and synergistic artificial intelligence for software engineering: Vision and roadmaps. *CoRR*, abs/2309.04142, 2023.