Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead

Xin Zhou[†], Sicong Cao[‡], Xiaobing Sun[‡], and David Lo[†]

[†]School of Computing and Information Systems, Singapore Management University

Singapore

[‡]School of Information Engineering, Yangzhou University

China

xinzhou.2020@phdcs.smu.edu.sg,davidlo@smu.edu.sg

{Dx120210088,xbsun}@yzu.edu.cn

ABSTRACT

The significant advancements in Large Language Models (LLMs) have resulted in their widespread adoption across various tasks within Software Engineering (SE), including vulnerability detection and repair. Numerous recent studies have investigated the application of LLMs to enhance vulnerability detection and repair tasks. Despite the increasing research interest, there is currently no existing survey that focuses on the utilization of LLMs for vulnerability detection and repair. In this paper, we aim to bridge this gap by offering a systematic literature review of approaches aimed at improving vulnerability detection and repair through the utilization of LLMs. The review encompasses research work from leading SE, AI, and Security conferences and journals, covering 36 papers published at 21 distinct venues. By answering three key research questions, we aim to (1) summarize the LLMs employed in the relevant literature, (2) categorize various LLM adaptation techniques in vulnerability detection, and (3) classify various LLM adaptation techniques in vulnerability repair. Based on our findings, we have identified a series of challenges that still need to be tackled considering existing studies. Additionally, we have outlined a roadmap highlighting potential opportunities that we believe are pertinent and crucial for future research endeavors.

1 INTRODUCTION

A software vulnerability refers to a flaw or weakness in a software system that can be exploited by attackers. Recently, the number of software vulnerabilities has increased significantly [71], affecting numerous software systems. To mitigate these issues, researchers have proposed methods for automatic detection and repair of identified vulnerabilities. However, traditional techniques, such as rule-based detectors or program analysis-based repair tools, encounter challenges due to high false positive rates [66] and their inability to work for diverse types of vulnerabilities [94], respectively.

Recently, Large Language Models (LLMs) pre-trained on large corpus have demonstrated remarkable effectiveness across various natural language and software engineering tasks [30]. Given their recent success, researchers have proposed various LLM-based approaches to improve automated vulnerability detection and repair, demonstrating promising outcomes for both detection and repair tasks [68, 94]. LLM-based approaches for vulnerability detection and repair are increasingly attracting attention due to their potential to automatically learn features from known vulnerabilities and find/fix unseen ones. Furthermore, LLMs have the potential to utilize rich knowledge acquired from large-scale pre-training to enhance vulnerability detection and repair.

Despite the increasing research interest in utilizing LLMs for vulnerability detection and repair, to the best of our knowledge, there has been no comprehensive literature review summarizing on the state-of-the-art approaches, current challenges, and future directions in this field. To effectively chart the most promising path forward for research on the utilization of LLM techniques in vulnerability detection and repair, we conducted a **Systematic Literature Review (SLR)** to bridge this gap, providing valuable insights to the community. In this paper, we collected **36** primary studies over the last 6 years (2018-2024). We then summarized the LLMs used by relevant papers and classified various techniques used to adapt LLMs. We also discussed the key challenges associated with using LLMs for vulnerability detection and repair and proposed multiple potential research directions.

Related Literature Reviews. Researchers have undertaken a series of research endeavors concerning Machine Learning (ML) for source code vulnerability detection or repair [23, 30, 39, 82, 90]. Hou et al. [30] conducted a systematic literature review on LLMs for SE. However, due to the extensive range of SE tasks to summarize, their review did not categorize the detailed utilization of LLMs in vulnerability detection and repair. In contrast, our literature review is more focused on vulnerability detection and repair. Ghaffarian et al. [23] studied vulnerability analysis and discovery approaches published till 2016. Lin et al. [39] reviewed vulnerability detection approaches utilizing deep learning till 2020. Wu et al. [82] investigated vulnerability detection approaches published before May 2022. However, their focus was not on LLMs and primarily covered general ML models such as recurrent neural networks. In contrast, our literature review includes papers published until March 2024 and focuses on LLMs, encompassing 15 distinct LLMs utilized in 36 relevant studies. Recently, Zhang et al. [90] reviewed learning-based automated program repair, which covered vulnerability repair as part of its study scope. In Different from Zhang et al., our literature review is more focused on LLMs and vulnerability repair: we included more recent studies, offered a detailed categorization of LLM usages in vulnerability repair, and discussed specialized future directions for vulnerability repair with LLMs.

In general, this study makes the following **contributions**:

- We present a systematic review of recent **36** primary studies focusing on the utilization of LLMs for vulnerability detection and repair.
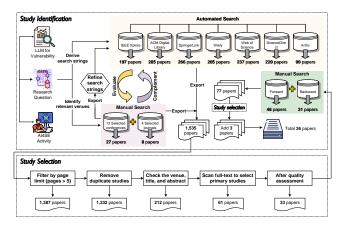
Figure 1: Study Identification and Selection Process

- We offer a comprehensive summary of the LLMs utilized in the relevant literature and categorize various techniques used to adapt LLMs to do these two tasks (vulnerability detection and repair).
- We discuss the key challenges associated with using LLMs for vulnerability detection and repair, and propose multiple potential research directions.

2 PRELIMINARIES

2.1 Vulnerability Detection/Repair Formulation

In this study, we focus on source code vulnerability detection and repair. Here, we present the task formulations of these two tasks.

Vulnerability Detection. Vulnerability detection is typically framed as a binary classification task: $X_i \rightarrow Y_i$. Specifically, given an input source code function X_i, a model predicts whether the input function is vulnerable ($Y_i = 1$) or non-vulnerable ($Y_i = 0$).

Vulnerability Repair. Vulnerability repair studies using LLMs frame the task as a sequence-to-sequence problem: $X_i \rightarrow Y_i$. Specifically, given a vulnerable code snippet X_i, an LLM model generates the corresponding repaired code Y_i.

2.2 Large Language Models (LLMs)

The term Large Language Model (LLM) was introduced to distinguish language models based on their parameter size, specifically referring to large-sized pre-trained language models [93]. However, the literature lacks a formal consensus on the minimum parameter scale for LLMs [76]. In this paper, we adopt the LLM scope division and taxonomy introduced by Pan et al. [56] and categorize the mainstream LLMs into three groups according to their architectures: 1) encoder-only, 2) encoder-decoder, and 3) decoder-only LLMs. We will provide a brief introduction to some representative LLMs for each category due to the space limit.

Encoder-only LLMs. Encoder-only LLMs are a type of neural network architecture that utilizes only the encoder component of the Transformer model [18]. In the SE domain, examples of encoder-only LLMs include CodeBERT [19], GraphCodeBERT [26], CuBERT [33], VulBERTa [27], CCBERT [96], SOBERT [29], and BERTOverflow [69].

Encoder-decoder LLMs. Encoder-decoder LLMs integrate both the encoder and decoder modules of the Transformer model [73].

The encoder processes the input sentence, while the decoder generates the target output text/code. Prominent examples of encoder-decoder LLMs include PLBART [9], T5 [62], CodeT5 [78], UniXcoder [25], and NatGen [12].

Decoder-only LLMs. Decoder-only LLMs exclusively utilize the decoder module of the Transformer model to generate the target output text/code. The GPT series, including GPT-2 [61], GPT-3 [11], GPT-3.5 [54], and GPT-4 [55], stand as prominent implementations of this model series. Additionally, in the SE domain, there are numerous decoder-only LLMs specialized for code as well. Examples include CodeGPT [44], Codex [13], Polycoder [84], Incoder [20], CodeGen series [51, 53], Copilot [24], Code Llama [46], and Star-Coder [37].

3 REVIEW METHODOLOGY

3.1 Research Question

In this paper, we focus on investigating the following three **Research Questions (RQs)**:

- **RQ1: What LLMs have been utilized to solve vulnerability detection and repair tasks?**
- **RQ2: How are LLMs adapted for vulnerability detection?**
- **RQ3: How are LLMs adapted for vulnerability repair?**

3.2 Search Strategy

As shown in Fig. 1, following the guide by Zhang et al. [88], our initial step is to identify primary studies to answer the research questions (RQs) above. Because the first LLM (i.e., BERT [18]) was introduced in 2018, our search focused on papers published from 2018 onwards (i.e., from January 1st, 2018, to March 1st, 2024). Next, we identified the top peer-reviewed and influential conference and journal venues in the domains of SE, AI, and Security. We included 13 conferences (ICSE, ESEC/FSE, ASE, ISSTA, CCS, S&P, USENIX Security, NDSS, AAAI, IJCAI, ICML, NIPS, ICLR) and 4 journals (TOSEM, TSE, TDSC, TIFS). After the manual searching, we identified 35 papers that were relevant to our research objectives.

In addition to manually searching primary studies from top-tier venues, we also conducted an automated search across 7 popular databases, including IEEE Xplore [3], ACM Digital Library [1], SpringerLink [5], Wiely [7], ScienceDirect [4], Web of Science [6] and arXiv [2]. The search string used in the automated search is crafted from the relevant papers identified in the manual search. Please kindly check the complete set of search keywords in our online appendix [8] due to the limited space. After conducting the automatic search, we collected 1,500 relevant studies with the automatic search from these 7 popular databases.

3.3 Study Selection

Inclusion and Exclusion Criteria. After the paper collection, we conducted a relevance assessment according to the following inclusion and exclusion criteria:

- ✓ *The paper must be written in English.*
- ✓ *The paper must have an accessible full text.*
- ✓ *The paper must be a peer-reviewed full research paper published either in a conference proceeding or a journal.*
- ✓ *The paper must adopt LLM techniques to solve source code vulnerability detection or repair.*
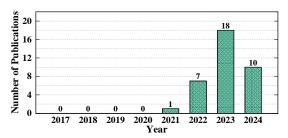
Figure 2: Distribution of Publications per Year

✗ *The paper has less than 5 pages.*

✗ *Books, keynote records, panel summaries, technical reports, theses, tool demos papers, editorials, or venues not subject to a full peer-review process.*

✗ *The paper is a literature review or survey.*

✗ *Duplicate papers or similar studies authored by the same authors.*

✗ *The paper does not utilize LLMs, e.g., using graph neural networks.*

✗ *The paper mentions LLMs only in future work or discussions rather than using LLMs in the approach.*

✗ *The paper does not involve source code vulnerability detection or repair tasks.*

In the first phase, by filtering out short papers (exclusion criteria 1) and deduplication (exclusion criteria 4), the total number of included papers was reduced to 1,332. In the second phase, we manually examined the venue, title, and abstracts of the papers, and the total number of included papers declined to 212. Books, keynote records, panel summaries, technical reports, theses, tool demo papers, editorials, literature reviews, or survey papers were also discarded in this phase (exclusion criteria 2-3). In the third phase, we manually read the full text of the paper to remove irrelevant papers. Specifically, the vulnerability detection or repair papers, which do not utilize LLMs but other methods, e.g., graph neural networks (GNN) or recurrent neural networks (RNN), were dropped (exclusion criteria 5). We also excluded studies that do not focus on source code vulnerability, such as those on binary code, protocols, or network communication. Furthermore, we removed studies that just discussed LLM as an idea or future work (exclusion criteria 6). We also removed the papers focusing on other tasks rather than vulnerability detection or repair, such as vulnerable data generation, vulnerability assessment, etc. (exclusion criteria 7). After the third phase, we identify 61 primary studies directly relevant to our research topic.

Quality Assessment. To prevent biases introduced by low-quality studies, we formulated five Quality Assessment Criteria (see our online appendix [8]) to assess the clarity, validity, and significance of the papers. We employed a scoring system ranging from 0 to 3 (poor, fair, good, excellent) for each quality assessment criterion. Following the manual assignment of scores, we selected papers with total scores exceeding 12 (80% of the maximum possible score). After this quality assessment, we obtained 33 papers.

Forward and Backward Snowballing. To avoid omitting any possibly relevant work during our manual and automated search process, we also performed lightweight backward and forward snowballing [81]. This involved reviewing both the references cited in our selected 33 primary studies and the publications that cited these studies. As a supplement, we gathered 77 more papers and
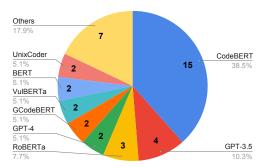

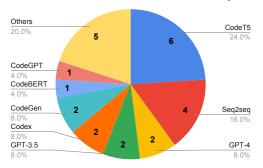**Figure 3: Distribution of LLMs for Vulnerability Detection**

Figure 4: Distribution of LLMs for Vulnerability Repair

repeated the entire study selection process, including filtering, deduplication, and quality assessment, which resulted in the identification of three additional papers. *Thus, we obtained a final set of **36** papers to study.*

3.4 Data Extraction and Analysis

Fig. 2 illustrates the distribution of selected primary studies across each year. The earliest relevant study we identified was published in 2021 [99]. Subsequently, interest in exploring LLMs for vulnerability detection and repair has steadily increased, peaking in 2023, with 50% of the total studied papers. Additionally, it is noteworthy that in the first two months of 2024 alone, the number of published papers has already reached 10, constituting 27.8% of the total papers. This suggests a growing research interest in leveraging LLMs for vulnerability detection and repair. We also analyzed the publication venues of the included studies. ICSE emerges as the predominant conference venue for LLM studies on vulnerability detection and repair, contributing 27.8% of the total number of studies. Other notable venues are TSE (8.3%), FSE (5.6%), EMSE (5.6%), ISSTA (5.6%), SCAM (5.6%), and TOSEM (2.8%).

4 RQ1: WHAT LLMS HAVE BEEN UTILIZED?

In general, out of the 36 included studies, we identified 15 distinct LLMs that have been utilized.

LLMs Used for Vulnerability Detection. Fig. 3 illustrates the distribution of the LLMs utilized for vulnerability detection. Code-BERT [19] emerges as the predominant LLM in addressing vulnerability detection to date, representing 38.5% (15/39) of the use of LLMs in the included studies. Following closely, GPT-3.5 becomes the second most frequently studied model, accounting for 10.3% (4/39). Regarding the categories of LLMs, encoder-only LLMs comprise 61.5% (24/39) of the use of LLMs in the included studies.

Figure 5: Adaptation Techniques of LLMs for Vul. Detection

Following that, decoder-only LLMs account for 28.2% (11/39), and encoder-decoder LLMs constitute 10.3% (4/39).

LLMs Used for Vulnerability Repair. Fig. 4 illustrates the distribution of LLMs utilized for vulnerability repair. Unlike the detection task, CodeT5 [78] emerges as the predominant LLM in addressing vulnerability repair to date, representing 24.0% (6/25) of the use of LLMs in the included studies. Following closely, domain-specific pre-trained Seq2Seq Transformers become the second most frequently studied model, accounting for 16.0% (4/25) of the use of LLMs. Regarding the categories of LLMs, encoder-decoder LLMs comprise 48% (12/25) of LLMs used. Following that, decoder-only LLMs and encoder-only LLMs constitute 44% (11/25) and 8% (2/25) of the use of LLMs, respectively.

> **Answer to RQ1**: Our analysis indicates that, to date, encoder-only LLMs have dominated vulnerability detection, while encoder-decoder LLMs have been prominent for vulnerability repair.

5 RQ2: HOW ARE LLMS ADAPTED FOR VULNERABILITY DETECTION?

In RQ2, we shift our focus to examining the specific adaptation techniques of LLMs in vulnerability detection. Regarding the usage of LLMs, we summarize ***three major categories*** from the included studies: **1) fine-tuning** [18], which updates the parameters of LLMs using a labeled dataset, **2) zero-shot prompting** [11], which freezes the model parameters and does not consider any labeled data, and **3) few-shot prompting** [11], which also freezes the parameters but considers a few labeled example data points. As depicted in Fig. 5, 82% of the studies utilize fine-tuning, while 11% and 7% employ zero-shot prompting and few-shot prompting, respectively.

In the following subsection, we will introduce detailed categories of LLM adaptation techniques for each of the following: *1) fine-tuning, 2) zero-shot prompting*, and *3) few-shot prompting*.

5.1 How are LLMs adapted for Vul. Detection?

5.1.1 ***Fine-tuning****.* The fine-tuning process usually involves several steps/stages, such as data preparation, model design, model training, and model evaluation. Regarding fine-tuning, we classify adaptation techniques into five groups based on the stages they mainly target: *Data-centric innovations* (data preparation), *Combination with program analysis* (data preparation), *LLM+ other deep learning modules* (model design), *Domain-specific pre-training* (model training), and *Causal learning* (training optimization).

Data-centric Innovations. Data-centric innovations focus on optimizing the vulnerability detection data used for fine-tuning LLMs. Prior studies [14, 17, 34, 50, 86] have found that existing vulnerability detection data can suffer from imbalanced label distribution, noisy or incorrect labels, and scarcity of labeled data. Researchers [34, 80, 86] have explored how to address the data issues.

- *Data Sampling*: To address label imbalance, i.e., having more non-vulnerable code samples than vulnerable ones in the dataset, Yang et al. [86] applied various data sampling techniques. They found that random oversampling on raw code data enhances the ability of the LLM-based vulnerability detection approach to learn real vulnerable patterns.
- *Positive and Unlabeled Learning*: To address label quality issues such as noisy or incorrect labels, Wen et al. [80] proposed PILOT, which learns solely from positive (vulnerable) and unlabeled data for vulnerability detection. Specifically, PILOT generates pseudo-labels for selected unlabeled data and mitigates the data noise by using a mixed-supervision loss.
- *Counterfactual Training*: To enhance the diversity of labeled data, Kuang et al. [34] proposed perturbing user-defined identifiers in the source code while preserving the syntactic and semantic structure. This approach generates diverse counterfactual training data, which refers to hypothetical data (e.g., data after perturbing identifiers) differing from actual data (i.e., data without perturbation), useful for analyzing the effect of certain factors. Incorporating these counterfactual data enriches the training data for LLMs.

Combination of LLM with Program Analysis. Many LLMs undergo pre-training on extensive datasets through unsupervised objectives like masked language modeling [18] or next token prediction [61]. For those LLMs, they may prioritize capturing sequential features and could potentially overlook certain structural aspects crucial for understanding code. To address this limitation, several studies have proposed integrating program analysis techniques with LLMs. The idea involves utilizing program analysis to extract structural features/relations within code, which are then incorporated into LLMs to enhance their understanding. Specifically, Liu et al. [42] leveraged Joern [85] to build the AST and PDG of the function, leveraging this data to pre-train their LLM to predict statement-level control dependencies and token-level data dependencies within the function. Peng et al. [58] utilized program slicing to extract control and data dependency information, aiding LLMs in vulnerability detection. Wang et al. [75] proposed to learn program presentations by feeding static source code information and dynamic program execution traces with LLMs. Additionally, Zhang et al. [89] proposed decomposing syntax-based Control Flow Graphs

(CFGs) into multiple execution paths and feeding these paths to LLMs for vulnerability detection.

Combination of LLM with Other Deep Learning Modules. LLMs have their own inherent limitations. Firstly, most LLMs are based on the Transformer architecture, which primarily models sequential relations and features. Secondly, some LLMs (e.g., Code-BERT) impose restrictions on the length of input code snippets. For instance, the most frequently used LLM in vulnerability detection, CodeBERT, can only process 512 tokens. To address these limitations, researchers [70, 99] have attempted to combine LLMs with other DL modules:

- *LLM+GNN*: To leverage the structural features of code more effectively, Tang et al. [70] introduced CSGVD, which employs graph neural networks (GNN) to extract the graph features of code and combine them with the features extracted by CodeBERT.

- *LLM+Bi-LSTM*: To address the length constraints of LLMs, Ziems et al. [99] first segmented the input code into multiple fixed-size segments. They then utilized BERT to encode each segment and incorporated a Bidirectional Long Short-Term Memory (Bi-LSTM) module to process the output of BERT on each segment. Finally, a softmax classifier was applied to the last hidden state of the Bi-LSTM to produce the final classification scores.

Domain-specific Pre-training. Domain-specific pre-training involves pre-training an LLM on data specific to a particular domain, such as vulnerability data, before fine-tuning it for a specific task within that domain. This process enables the LLM to better understand the data relevant to the domain. Several studies in the field of vulnerability detection have adopted this technique [27, 42, 49, 75]. These studies typically use one of three pre-training objectives:

- *Masked Language Modeling*: This pre-training objective involves training the LLM to predict masked tokens in a corrupted code snippet. Hanif and Mahmood [27] introduced VulBERTa, which pre-trained a RoBERTa [40] model on open-source C/C++ projects using the Masked Language Modeling objective. Since C/C++ is the programming language of the vulnerability detection data used to evaluate VulBERTa, this pre-training enhances VulBERTa's ability to understand C/C++ code.

- *Contrastive Learning:* This pre-training objective is to minimize the distance between similar functions while maximizing the distance between dissimilar functions. Through contrastive learning, the LLM can learn to capture the distinctive features of code. In particular, Ni et al. [49] and Wang et al. [75] utilized different hidden dropout masks to transform the same input function into positive samples (i.e., those similar to the input function) while regarding other distinct functions as dissimilar samples. Then they pre-trained an LLM following the contrastive learning objective.

- *Predicting Program Dependencies:* This pre-training objective aims to enhance the LLM by guiding the model to learn the knowledge necessary for analyzing dependencies in programs. Specifically, Liu et al. [42] pre-trained an encoder-only LLM on code from open-source C/C++ projects to predict the statement-level control dependency and token-level data dependency.

After the pre-training, those domain-specific pre-trained LLMs undergo fine-tuning on the vulnerability detection dataset to perform vulnerability detection.
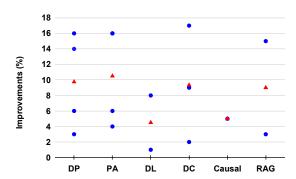

Figure 6: Effectiveness of Adaptation Techniques for Vulnerability Detection

Causal Learning. Although promising, LLMs have been found to lack robustness under perturbation or when encountering out-of-distribution (OOD) data [45]. Rahman et al. [45] suggested that this weak robustness may be attributed to LLMs learning non-robust features, such as variable names, that have spurious correlations with labels. To address this issue, Rahman et al. proposed CausalVul, which first designs perturbations to identify spurious features and then applies causal learning algorithms, specifically do-calculus, on top of LLMs to promote causal-based prediction, enhancing the robustness of LLMs in vulnerability detection.

5.1.2 Zero-shot Prompting. For zero-shot prompting, we only observe one adaptation technique (i.e., prompt engineering).

Prompt Engineering. Researchers [22, 59, 97] have attempted to devise effective prompts to guide LLMs in conducting vulnerability detection in the zero-shot prompting manner. Specifically, Zhou et al. [97] devised a prompt: *"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 [code]. Let's start:".* Fu et al. [22] proposed the prompt: *"Predict Whether the C/C++ function below is vulnerable. Strictly return 1 for a vulnerable function and 0 for a non-vulnerable function without further explanation."* Purba et al. [59] devised a prompt: *"[code] Is this code vulnerable? Answer in only Yes or No".* In these prompts, *"[code]"* refers to the input code.

5.1.3 Few-shot Prompting. For few-shot prompting, we only find one adaptation technique (i.e., retrieval augmentation).

Retrieval Augmentation. Retrieval augmentation is a technique used to enhance the few-shot prompting. It involves retrieving similar labeled data samples from the training set when given a test data sample and using these retrieved data as examples to guide the prediction of LLMs on the test sample. Liu et al. [41] proposed using efficient retrieval tools such as BM-25 and TF-IDF for retrieval. Zhou et al. [97] suggested using CodeBERT as a retrieval tool. This method first transforms code snippets into semantic vectors and then quantifies the similarity between two code snippets by calculating the Cosine similarity of their respective semantic vectors. It finally returned the top similar code based on the similarity scores.

5.2 How effective are the techniques used?

In this subsection, we examine the effectiveness of LLM adaptation techniques utilized in vulnerability detection. Specifically, we conduct a manual review of the ablation studies in the primary research

to compare the performance of approaches with and without those LLM adaptation techniques. The differences observed can indicate the effectiveness of the LLM adaptation techniques. If a study does not provide such information, we exclude the results of the paper from this subsection.

Fig. 6 illustrates the effectiveness of LLM adaptation techniques in vulnerability detection. The blue dots represent the improvement rates reported in the primary studies, while the red triangle denotes the average improvement across the studies. When fine-tuning an LLM, *domain-specific pre-training (DP), combination with program analysis (PA), combination with other DL modules (DL), data-centric innovations (DC)*, and *causal learning (Causal)* lead to improvements ranging from 3% to 16%, 4% to 16%, 1% to 8%, 2% to 17%, and 5%, respectively, in terms of Accuracy scores. For zero-shot prompting, we did not find a clear ablation study demonstrating the contribution of prompt engineering. For few-shot prompting, *retrieval augmentation (RAG)* has enhanced the effectiveness of LLMs, improving accuracy from 3% to 15%. These findings highlight the extent these methods were effective as reported in prior studies.

> **Answer to RQ2**: Our analysis revealed three commonly used techniques to adapt LLMs for vulnerability detection: including *fine-tuning* (≈82%), *zero-shot prompting* (≈11%), and *few-shot prompting* (≈7%). We noted that different adaptation techniques led to varying improvements, with *combination with program analysis (PA)* technique achieving the highest average improvement.

6 RQ3: HOW ARE LLMS ADAPTED FOR VULNERABILITY REPAIR?

Regarding the usage of LLMs, we summarize **three major categories** from the included studies: **1) fine-tuning** (approximately 80%), which updates the parameters of LLMs using a labeled dataset, **2) zero-shot prompting** (approximately 13%), which freezes the parameters of LLMs and does not consider any labeled data, and **3) few-shot prompting** (approximately 7%), which also freezes the parameters of LLMs but considers a few labeled examples.

In the following subsection, we will introduce detailed categories of LLM adaptation techniques for each of the following: *1) fine-tuning, 2) zero-shot prompting*, and *3) few-shot prompting*.

6.1 How are LLMs adapted for Vul. Repair?

*6.1.1 **Fine-tuning**.* The fine-tuning process usually involves several steps/stages, such as data preparation, model design, and model training. Regarding fine-tuning, we categorize adaptation techniques into four groups based on the stages they mainly target: *Diverse relevant inputs* (data preparation), *Model-centric innovations* (model design), *Domain-specific pre-training* (model training), and *Reinforcement learning* (training optimization).

Diverse Relevant Inputs. Several studies have shown that in addition to the input vulnerable code, incorporating other diverse relevant inputs can boost the effectiveness of LLMs. Specifically, the AST of the vulnerable code input [98], vulnerability descriptions [79, 98], and vulnerable code examples shared on CWE websites [94] can enhance the LLMs in repairing vulnerability. In addition, Wei et al. [79] found that vulnerability-inducing commits and vulnerability-fixing commits can also improve the effectiveness of
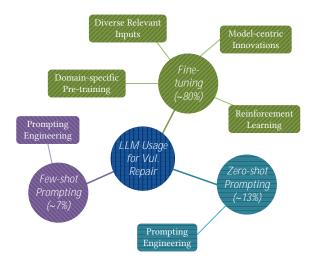

Figure 7: Adaptation Techniques of LLMs for Vul. Repair

the model. Lastly, Zhou et al. [94] found that certain vulnerable functions exceed the length limit of Transformer-based LLMs like CodeT5 (512 subtokens) [94]. To address this limitation, they applied the Fusion-in-Decoder framework to partition a long code function into multiple segments and feed those segments into LLMs one by one. Their findings demonstrated that those segments can boost model effectiveness.

Model-Centric Innovations. Model-centric innovations encompass methodologies that prioritize revising the model architecture of LLMs (i.e., the Transformer [73]). Specifically, Fu et al. [21] drew inspiration from Vision Transformer (VIT)-based object detection techniques in computer vision. They proposed vulnerability queries within the pre-trained Transformer model to identify vulnerable code blocks within the source code. Additionally, they trained a model to learn a vulnerability mask, enhancing the attention of the vulnerability queries on the vulnerable code areas.

Domain-specific Pre-training. Domain-specific pre-training involves pre-training an LLM on data specific to a particular domain before fine-tuning it for the target task. Considering the similarity between bug-fixing and vulnerability-fixing tasks, several studies [15, 16, 91, 94] considered the task of bug-fixing as a pre-training task to enhance LLMs. Specifically, they first pre-trained LLMs on a bug fix corpus to fix bugs, and then fine-tuned LLMs on a vulnerability fix dataset to repair vulnerabilities. This kind of pre-training technique can be considered as *Transfer Learning*.

Reinforcement Learning. Islam et al. [31] introduced Secure-Code, an LLM tuned using a reinforcement learning framework. This approach integrated syntactic and semantic rewards to generate fixes for vulnerable code. Specifically, they leveraged the CodeBLEU [63] score as the syntactic reward and BERTScore [92] as the semantic reward. After combining these rewards, they applied the Proximal Policy Optimization (PPO) algorithm [64] to fine-tune the CodeGen2-7B [52] model.

*6.1.2 **Zero-shot and Few-shot Prompting**.* For both zero-shot and few-shot prompting, we only observe one adaptation technique.

Prompt Engineering. For zero-shot prompting, Pearce et al. [57] devised various prompts to guide LLMs in repairing vulnerabilities in a zero-shot manner. They incorporated error messages from
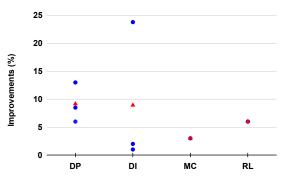
Figure 8: Effectiveness of Adaptation Techniques for Vulnerability Repair

static analysis tools like CodeQL or C sanitizers into their prompts, providing contextual information about the vulnerabilities. For instance, one prompt format is: "// BUG: [vulnerability-description] //MESSAGE: [error-message] [Code] //FIXED VERSION:", where [Code] represents the input vulnerable function. In addition, Wu et al. [83] also explored diverse prompts for directing LLMs in repairing vulnerabilities in a zero-shot manner. For example, they proposed a prompt that begins by commenting on the buggy code lines and adds "BUG:" to highlight them. Then, the prompt appends "FIXED:" at the end to guide the LLM in generating a corrected version of the code.

Furthermore, for few-shot prompting, Fu et al. [22] investigated a straightforward prompt strategy involving the provision of three repair examples within each prompt. This approach aimed to aid GPT-3.5 and GPT-4 in performing the repair task more effectively.

6.2 How effective are the techniques used?

Fig. 8 illustrates the effectiveness of LLM adaptation techniques in vulnerability repair. The blue dots represent the improvement rates reported in the primary studies, while the red triangle denotes the average improvement across the studies. When fine-tuning an LLM, *domain-specific pre-training (DP)*, *combining LLM with diverse relevant inputs (DI)*, *model-centric innovations (MC)*, and *reinforcement learning (RL)* have resulted in improvements ranging from 6% to 13%, 1% to 24%, 3% to 3.3%, and 6%, respectively, in terms of Accuracy scores. For zero-shot prompting and few-shot prompting, we did not find a clear ablation study demonstrating the contribution of prompt engineering. These results highlight the extent these methods were effective as reported in prior studies.

Answer to RQ3: Our analysis revealed three commonly used techniques to adapt LLMs for vulnerability repair: including *fine-tuning* (≈80%), *zero-shot prompting* (≈13%), and *few-shot prompting* (≈7%). We found that different adaptation techniques resulted in varying degrees of improvement, with *domain-specific pre-training (DP)* achieving the highest average improvement.

---

7 THE ROAD AHEAD

In this section, we discuss current challenges (Section 7.1) and highlight promising opportunities (Section 7.2) for applying LLMs techniques in vulnerability detection and repair.

7.1 Challenges

Challenge 1: Lack of High-quality Vulnerability Dataset. One major challenge is the lack of high-quality vulnerability benchmarks. Previous studies [14, 17, 50] have highlighted issues with existing vulnerability data, including noisy or incorrect labels (e.g., labeling clean code as vulnerable). This data quality issue is primarily attributed to the use of automatic vulnerability collection [50], which can gather large enough data for training DL-based models including LLMs but cannot ensure the complete correctness of the labels. While manually checking each data sample can ensure high quality, it is a very tedious and expensive process, especially when aiming for a large dataset. *Constructing a high-quality vulnerability benchmark remains an open challenge to date.* Moreover, there is a growing concern that the capabilities of LLMs may derive from the inclusion of evaluation datasets in the pre-training corpus of LLMs, a phenomenon known as data contamination [32]. To mitigate this concern, high-quality vulnerability benchmarks are preferred to have no overlap with the pre-training corpus of LLMs.

Challenge 2: Complexity in Vulnerability Data. Vulnerabilities can be inherently complicated, which brings challenges for their detection and repair with LLMs. For instance, inter-procedural vulnerabilities are prevalent in vulnerability data, and Li et al. [38] discovered that detecting inter-procedural vulnerabilities poses greater challenges than intra-procedural ones. Moreover, vulnerabilities encompass a wide array of Common Weakness Enumeration (CWE) types [95], but LLMs may struggle with less frequent CWE types compared to frequent types [94, 95]. In addition, vulnerabilities are typically represented in terms of code units, such as code lines, functions, or program slices within which the vulnerabilities occur. Sejfia et al. [65] observed a significant accuracy drop when detecting vulnerabilities that span multiple code units, such as spanning multiple functions. *Future research should consider the complex nature of vulnerabilities when designing LLM-based solutions.*

Challenge 3: Narrow Scope of Input Programs. Currently, LLM-based vulnerability detection and repair solutions primarily target the function level (or more fine-grained, at the line level). However, this narrow focus indicates that these approaches may not perform optimally when presented with a wider range of programs, such as classes or a whole repository. Function-level vulnerability detection approaches could overlook vulnerabilities that span multiple functions or classes [65]. Similarly, function-level vulnerability repair approaches fall short when tasked with modifications across multiple functions within the repository [94]. *In addition to functions, future research could propose LLM-based detection/repair approaches capable of handling a broader range of programs.*

Challenge 4: High Accuracy and Robustness. A vulnerability detection or repair solution with high accuracy is generally preferred, as it boosts developers' confidence in the reliability of detections and fixes. However, current state-of-the-art approaches [42, 94] have not yet achieved satisfactory accuracy, with 67.6% and 20% accuracy scores for vulnerability detection and repair, respectively. Moreover, the solution should maintain robustness against data perturbations or adversarial attacks to ensure its resilience. However, Yang et al. [87] and Rahman et al. [45] discovered that LLMs are not robust against data perturbations. *Future research should seek ways to improve the accuracy and robustness of LLM-based solutions.*

Challenge 5: Establishing Trust and Synergy with Developers. To enhance developers' trust, LLM-based approaches should explicitly notify developers when unable to provide accurate outputs, which allows developers to avoid spending time reviewing potentially inaccurate outputs [35]. In addition, there is limited interaction between developers and LLM-based vulnerability detection/repair solutions, which may hinder the establishment of trust and synergy during practical application. *To bridge this gap, future research should investigate more effective strategies for fostering collaboration and trust between developers and LLM-based solutions [43].* By nurturing trust and synergy, LLM-based solutions have the potential to evolve into intelligent allies, offering enhanced support to developers.

7.2 Opportunities

Opportunity 1: Curating High-quality Test Set. The absence of a high-quality vulnerability dataset poses a significant obstacle to vulnerability detection. While obtaining fully correct labels for a large dataset is expensive, a viable solution is to curate a high-quality test set (which is much smaller than the whole dataset) that can accurately assess progress in vulnerability detection. An easy approach for future work is to combine the manually checked vulnerability data samples scattered across several separate research works (e.g., [17, 28, 36]) to form a high-quality test set. Future works can then do an empirical study to recognize the real progress in vulnerability detection with the high-quality test set. Furthermore, the community can maintain a living high-quality test set by adding new manually verified data to it. This curated test set can serve as a reliable benchmark for vulnerability detection.

Opportunity 2: Repo-level Vulnerability Detection/Repair. Current vulnerability detection and repair techniques primarily focus on the function or line level. One key reason is the input length limitations (512 subtokens) of the small LLMs like CodeBERT and CodeT5, which are predominantly used in existing studies. The 512 subtokens limitation aligns well with the function level data but faces difficulty in scaling up to classes or repositories. However, the advent of recent larger LLMs with significantly higher input length capacity, such as Code Llama (16,384 subtokens) and GPT-4 (24,576 sub-tokens), enables processing repo-level data more effectively. This presents an opportunity for future research to explore repo-level vulnerability detection/repair by leveraging these larger LLMs.

Opportunity 3: Using Larger Decoder-only LLMs. As presented in RQ1, it is evident that researchers favor encoder-only language models (61.5%) for vulnerability detection and encoder-decoder models (48.0%) for repair. Moreover, only 6 out of the 36 studies investigate LLMs with more than 7 billion parameters, suggesting a focus on smaller LLMs. This preference contrasts with a broader trend observed in LLMs for SE, as highlighted in a recent literature review [30]. They found that since 2023, decoder-only models have become predominant, accounting for 73.1% of the papers studied in 2023. Additionally, among very large LLMs (e.g., >7B parameters), the majority are decoder-only models. Furthermore, decoder-only LLMs excel in generating code/text [55], making them well-suited for vulnerability repair, which is a generation task. While vulnerability detection is a classification task, it can be reframed as a generation task. For instance, decoder-only LLMs can be instructed to output *'it is vulnerable'* given a vulnerable code and *'it is non-vulnerable'* given a non-vulnerable code. Thus, large decoder-only LLMs can be leveraged to address both vulnerability detection and repair tasks. Currently, it seems that decoder-only LLMs, particularly those with parameters larger than 7 billion, are underutilized in vulnerability detection and repair, indicating untapped potential and offering opportunities for future work.

Opportunity 4: Advanced LLM Usage and Adaptation. Regarding LLMs usages, beyond the techniques observed in the included studies—such as *fine-tuning* and *zero/few-shot prompting*—there exist two more advanced usages of LLMs [48] that have not been explored yet in vulnerability detection and repair: 1) *LLM Agent*: LLMs can serve as agents to decompose complex tasks into smaller components and employ multiple LLMs to address them [77]; 2) *Usage of External Tools*: LLMs can utilize external tools such as search engines, external databases, and other resources to enhance them [60].

Regarding detailed LLMs adaptation techniques, as illustrated in Fig.5 and Fig.7, the majority of proposed adaptations in this field are designed for *fine-tuning*. However, a plethora of advanced adaptation techniques for *few-shot prompting* remains unexplored. These techniques include iterative retrieval augmentation [67], recursive retrieval augmentation [72], and adaptive retrieval augmentation [10]. Researchers can consider using those unexplored advanced LLM usages/adaptations in future works.

Opportunity 5: Customized LLMs for Vulnerability. Currently, widely used LLMs in vulnerability detection/repair are general-purpose LLMs (e.g., CodeBERT, CodeT5, and GPT-3.5) that do not fully exploit the wealth of open-source vulnerability data. A promising avenue is the development of customized LLMs tailored for vulnerability data. Some initial attempts in this direction include vulnGPT [74] and Microsoft Security Copilot [47]. However, as these solutions are proprietary, their customized LLM details may not be fully disclosed. We advocate for collaborative efforts to develop open-sourced and effective customized LLMs for vulnerability.

8 THREATS TO VALIDITY

The potential threat to validity is the risk of inadvertently excluding relevant studies during the literature search and selection phase. Incomplete summarization of keywords for vulnerability detection/repair or varied terminologies of LLMs may have caused relevant research studies to be missed in our review. To mitigate this risk, we initially performed a manual selection of 17 high-impact venues and extracted a relatively comprehensive set of standard keywords from relevant papers within these venues. In addition, we further augmented our search results by combining automated search with forward-backward snowballing.

9 CONCLUSION AND FUTURE WORK

The use of Large Language Models (LLMs) for vulnerability detection and repair has been garnering increasing attention. This paper presents a systematic literature review of 36 primary studies on LLMs for vulnerability detection and repair, sourced from leading SE, AI, and Security conferences and journals. This review begins by analyzing the types of LLMs used in primary studies, shedding light on researchers' preferences for different LLMs. Subsequently,

we categorized a variety of techniques for adapting LLMs. Through our analysis, this review also identifies the challenges in this field and proposes a research roadmap outlining promising avenues for future exploration. In the future, we plan to broaden this literature review by incorporating additional vulnerability-related tasks, such as vulnerability localization and vulnerability assessment. Our online appendix is available at [8].

REFERENCES

[1] [n. d.]. ACM Digital Library. https://dl.acm.org.
[2] [n. d.]. arXiv Database. https://arxiv.org.
[3] [n. d.]. IEEE Xplore Database. https://ieeexplore.ieee.org.
[4] [n. d.]. ScienceDirect Database. https://www.sciencedirect.com.
[5] [n. d.]. SpringerLink Database. https://link.springer.com.
[6] [n. d.]. Web of Science Database. https://www.webofscience.com.
[7] [n. d.]. Wiely Database. https://onlinelibrary.wiley.com.
[8] 2024. Online Appendix for This Review. https://docs.google.com/document/d/18-UrkfH35CNMGRjjsDYZGK6L1aC9wP3GsKCtrIekcUQ/edit?usp=sharing.
[9] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333* (2021).
[10] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511* (2023).
[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[12] Saikat Chakraborty, Toufique Ahmed, Yangruibo Ding, Premkumar T. Devanbu, and Baishakhi Ray. 2022. NatGen: generative pre-training by "naturalizing" source code. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, Abhik Roychoudhury, Cristian Cadar, and Miryung Kim (Eds.). ACM, 18–30. https://doi.org/10.1145/3540250.3549162
[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
[14] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David A. Wagner. 2023. DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2023, Hong Kong, China, October 16-18, 2023*. ACM, 654–668. https://doi.org/10.1145/3607199.3607242
[15] Zimin Chen, Steve Kommrusch, and Martin Monperrus. 2023. Neural Transfer Learning for Repairing Security Vulnerabilities in C Code. *IEEE Trans. Software Eng.* 49, 1 (2023), 147–165. https://doi.org/10.1109/TSE.2022.3147265
[16] Jianlei Chi, Yu Qu, Ting Liu, Qinghua Zheng, and Heng Yin. 2023. SeqTrans: Automatic Vulnerability Fix Via Sequence to Sequence Learning. *IEEE Trans. Software Eng.* 49, 2 (2023), 564–585. https://doi.org/10.1109/TSE.2022.3156637
[17] Roland Croft, Muhammad Ali Babar, and M. Mehdi Kholoosi. 2023. Data Quality for Software Vulnerability Datasets. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 121–133. https://doi.org/10.1109/ICSE48619.2023.00022
[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[19] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).
[20] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. InCoder: A Generative Model for Code Infilling and Synthesis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=hQwb-lbM6EL
[21] Michael Fu, Van Nguyen, Chakkrit Tantithamthavorn, Dinh Phung, and Trung Le. 2024. Vision Transformer-Inspired Automated Vulnerability Repair. *ACM Transactions on Software Engineering and Methodology* (2024).
[22] Michael Fu, Chakkrit Tantithamthavorn, Van Nguyen, and Trung Le. 2023. Chatgpt for vulnerability detection, classification, and repair: How far are we? *APSEC* (2023).
[23] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM computing surveys (CSUR)* 50, 4 (2017), 1–36.

[24] GitHub. 2023. Github copilot. https://copilot.github.com.
[25] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 7212–7225.
[26] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).
[27] Hazim Hanif and Sergio Maffeis. 2022. VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection. In *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*. IEEE, 1–8. https://doi.org/10.1109/IJCNN55064.2022.9892280
[28] Jingxuan He and Martin Vechev. 2023. Large language models for code: Security hardening and adversarial testing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1865–1879.
[29] Junda He, Zhou Xin, Bowen Xu, Ting Zhang, Kisub Kim, Zhou Yang, Ferdian Thung, Ivana Irsan, and David Lo. 2023. Representation Learning for Stack Overflow Posts: How Far are We? *arXiv preprint arXiv:2303.06853* (2023).
[30] Xinying Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John C. Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. *ArXiv* abs/2308.10620 (2023). https://api.semanticscholar.org/CorpusID:261048648
[31] Nafis Tanveer Islam and Peyman Najafirad. 2024. Code Security Vulnerability Repair Using Reinforcement Learning with Large Language Models. *AAAI Workshop* (2024).
[32] Minhao Jiang, Ken Ziyu Liu, Ming Zhong, Rylan Schaeffer, Siru Ouyang, Jiawei Han, and Sanmi Koyejo. 2024. Investigating Data Contamination for Pre-training Language Models. arXiv:2401.06059 [cs.CL]
[33] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2019. Learning and Evaluating Contextual Embedding of Source Code. In *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:220425306
[34] Hongyu Kuang, Feng Yang, Long Zhang, Gaigai Tang, and Lin Yang. 2023. Leveraging User-Defined Identifiers for Counterfactual Data Generation in Source Code Vulnerability Detection. In *23rd IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2023, Bogotá, Colombia, October 2-3, 2023*, Leon Moonen, Christian D. Newman, and Alessandra Gorla (Eds.). IEEE, 143–150. https://doi.org/10.1109/SCAM59687.2023.00024
[35] Tien-Duy B Le, David Lo, and Ferdian Thung. 2015. Should i follow this fault localization tool's output? automated prediction of fault localization effectiveness. *Empirical Software Engineering* 20 (2015), 1237–1274.
[36] Kaixuan Li, Sen Chen, Lingling Fan, Ruitao Feng, Han Liu, Chengwei Liu, Yang Liu, and Yixiang Chen. 2023. Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 921–933.
[37] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nourhan Fahmy, Urvashi Bhattacharyya, W. Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jana Ebert, Tri Dao, Mayank Mishra, Alexander Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean M. Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *ArXiv* abs/2305.06161 (2023). https://api.semanticscholar.org/CorpusID:258588247
[38] Zhen Li, Ning Wang, Deqing Zou, Yating Li, Ruqian Zhang, Shouhuai Xu, Chao Zhang, and Hai Jin. 2024. On the Effectiveness of Function-Level Vulnerability Detectors for Inter-Procedural Vulnerabilities. *ICSE* (2024).
[39] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. 2020. Software vulnerability detection using deep neural networks: a survey. *Proc. IEEE* 108, 10 (2020), 1825–1848.
[40] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[41] Zhihong Liu, Qing Liao, Wenchao Gu, and Cuiyun Gao. 2023. Software Vulnerability Detection with GPT and In-Context Learning. In *8th International Conference on Data Science in Cyberspace, DSC 2023, Hefei, China, August 18-20, 2023*. IEEE, 229–236. https://doi.org/10.1109/DSC59305.2023.00041

[42] Zhongxin Liu, Zhijie Tang, Junwei Zhang, Xin Xia, and Xiaohu Yang. 2024. Pre-training by Predicting Program Dependencies for Vulnerability Analysis Tasks. *ICSE* (2024).

[43] David Lo. 2023. Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps. *CoRR* abs/2309.04142 (2023). https://doi.org/10.48550/ARXIV.2309.04142 arXiv:2309.04142

[44] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664* (2021).

[45] Md Mahbubur Rahman, Ira Ceka, Chengzhi Mao, Saikat Chakraborty, Baishakhi Ray, and Wei Le. 2024. Towards Causal Deep Learning for Vulnerability Detection. *ICSE* (2024).

[46] Meta. 2023. Code Llama: Open Foundation Models for Code. https://ai.meta.com/research/publications/code-llama-open-foundation-models-for-code/.

[47] Microsoft. 2024. Microsoft Copilot for Security. https://microsoft.github.io/PartnerResources/skilling/microsoft-security-academy/microsoft-security-copilot.

[48] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large Language Models: A Survey. arXiv:2402.06196 [cs.CL]

[49] Chao Ni, Xin Yin, Kaiwen Yang, Dehai Zhao, Zhenchang Xing, and Xin Xia. 2023. Distinguishing Look-Alike Innocent and Vulnerable Code by Subtle Semantic Representation Learning and Explanation. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1611–1622.

[50] Xu Nie, Ningke Li, Kailong Wang, Shangguang Wang, Xiapu Luo, and Haoyu Wang. 2023. Understanding and Tackling Label Errors in Deep Learning-Based Vulnerability Detection (Experience Paper). In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, René Just and Gordon Fraser (Eds.). ACM, 52–63. https://doi.org/10.1145/3597926.3598037

[51] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. CodeGen2: Lessons for Training LLMs on Programming and Natural Languages. *CoRR* abs/2305.02309 (2023). https://doi.org/10.48550/ARXIV.2305.02309 arXiv:2305.02309

[52] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309* (2023).

[53] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=iaYcJKpY2B_

[54] OpenAI. 2022. GPT-3.5. https://platform.openai.com/docs/models/gpt-3-5.

[55] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[56] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2023. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *arXiv preprint arXiv:2306.08302* (2023).

[57] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2021. Examining zero-shot vulnerability repair with large language models. *arXiv preprint arXiv:2112.02125* (2021).

[58] Tao Peng, Shixu Chen, Fei Zhu, Junwei Tang, Junping Liu, and Xinrong Hu. 2023. PTLVD:Program Slicing and Transformer-based Line-level Vulnerability Detection System. In *23rd IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2023, Bogotá, Colombia, October 2-3, 2023*, Leon Moonen, Christian D. Newman, and Alessandra Gorla (Eds.). IEEE, 162–173. https://doi.org/10.1109/SCAM59687.2023.00026

[59] Moumita Das Purba, Arpita Ghosh, Benjamin J. Radford, and Bill Chu. 2023. Software Vulnerability Detection using Large Language Models. In *34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023 - Workshops, Florence, Italy, October 9-12, 2023*. IEEE, 112–119. https://doi.org/10.1109/ISSREW60843.2023.00058

[60] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. arXiv:2307.16789 [cs.AI]

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

[62] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[63] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a

[64] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[65] Adriana Sejfia, Satyaki Das, Saad Shafiq, and Nenad Medvidović. 2024. Toward Improved Deep Learning-based Vulnerability Detection. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.

[66] Hossain Shahriar and Mohammad Zulkernine. 2012. Mitigating program security vulnerabilities: Approaches and challenges. *ACM Computing Surveys (CSUR)* 44, 3 (2012), 1–46.

[67] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294* (2023).

[68] Benjamin Steenhoek, Md Mahbubur Rahman, Richard Jiles, and Wei Le. 2023. An empirical study of deep learning models for vulnerability detection. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2237–2248.

[69] Jeniya Tabassum, Mounica Maddela, Wei Xu, and Alan Ritter. 2020. Code and named entity recognition in stackoverflow. *arXiv preprint arXiv:2005.01634* (2020).

[70] Wei Tang, Mingwei Tang, Minchao Ban, Ziguo Zhao, and Mingjun Feng. 2023. CSGVD: A deep learning approach combining sequence and graph embedding for source code vulnerability detection. *J. Syst. Softw.* 199 (2023), 111623. https://doi.org/10.1016/J.JSS.2023.111623

[71] ED TARGETT. 2022. We analysed 90,000+ software vulnerabilities: Here's what we learned. https://www.thestack.technology/analysis-of-cves-in-2022-software-vulnerabilities-cwes-most-dangerous/.

[72] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509* (2022).

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

[74] Vicarius. 2023. 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/.

[75] Huanting Wang, Zhanyong Tang, Shin Hwei Tan, Jie Wang, Yuzhe Liu, Hejun Fang, Chunwei Xia, and Zheng Wang. 2024. Combining Structured Static Code Information and Dynamic Symbolic Traces for Software Vulnerability Prediction. In *Proceedings of the 46th International Conference on Software Engineering*. ACM.

[76] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2023. Software Testing with Large Language Model: Survey, Landscape, and Vision. *arXiv preprint arXiv:2307.07221* (2023).

[77] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 1–26.

[78] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[79] Ying Wei, Lili Bo, Xiaoxue Wu, Yue Li, Zhenlei Ye, Xiaobing Sun, and Bin Li. 2023. VulRep: vulnerability repair based on inducing commits and fixing commits. *EURASIP Journal on Wireless Communications and Networking* 2023, 1 (2023), 34.

[80] Xin-Cheng Wen, Xinchen Wang, Cuiyun Gao, Shaohua Wang, Yang Liu, and Zhaoquan Gu. 2023. When Less is Enough: Positive and Unlabeled Learning Model for Vulnerability Detection. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 345–357. https://doi.org/10.1109/ASE56229.2023.00144

[81] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 38:1–38:10.

[82] Bolun Wu, Futai Zou, et al. 2022. Code vulnerability detection based on deep sequence and graph models: A survey. *Security and Communication Networks* 2022 (2022).

[83] Yi Wu, Nan Jiang, Hung Viet Pham, Thibaud Lutellier, Jordan Davis, Lin Tan, Petr Babkin, and Sameena Shah. 2023. How Effective Are Neural Networks for Fixing Security Vulnerabilities. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, René Just and Gordon Fraser (Eds.). ACM, 1282–1294. https://doi.org/10.1145/3597926.3598135

[84] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *MAPS@PLDI 2022: 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022*, Swarat Chaudhuri and Charles Sutton (Eds.). ACM, 1–10. https://doi.org/10.1145/3520312.3534862

[85] Fabian Yamaguchi. 2023. Joern: A Source code analysis Tool. https://github.com/octopus-platform/joern.

[86] Xu Yang, Shaowei Wang, Yi Li, and Shaohua Wang. 2023. Does data sampling improve deep learning-based vulnerability detection? Yeas! and Nays!. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2287–2298. https://doi.org/10.1109/ICSE48619.2023.00192

[87] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural attack for pre-trained models of code. In *Proceedings of the 44th International Conference on Software Engineering*.

[88] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Inf. Softw. Technol.* 53, 6 (2011), 625–637.

[89] Junwei Zhang, Zhongxin Liu, Xing Hu, Xin Xia, and Shanping Li. 2023. Vulnerability Detection by Learning From Syntax-Based Execution Paths of Code. *IEEE Trans. Software Eng.* 49, 8 (2023), 4196–4212. https://doi.org/10.1109/TSE.2023.3286586

[90] Quanjun Zhang, Chunrong Fang, Yuxiang Ma, Weisong Sun, and Zhenyu Chen. 2023. A Survey of Learning-based Automated Program Repair. *arXiv preprint arXiv:2301.03270* (2023).

[91] Quanjun Zhang, Chunrong Fang, Bowen Yu, Weisong Sun, Tongke Zhang, and Zhenyu Chen. 2023. Pre-trained model-based automated software vulnerability repair: How far are we? *IEEE Transactions on Dependable and Secure Computing* (2023).

[92] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=SkeHuCVFDr

[93] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[94] Xin Zhou, Kisub Kim, Bowen Xu, DongGyun Han, and David Lo. 2024. Out of Sight, Out of Mind: Better Automatic Vulnerability Repair by Broadening Input Ranges and Sources. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 872–872.

[95] Xin Zhou, Kisub Kim, Bowen Xu, Jiakun Liu, DongGyun Han, and David Lo. 2023. The devil is in the tails: How long-tailed code distributions impact large language models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 40–52.

[96] Xin Zhou, Bowen Xu, DongGyun Han, Zhou Yang, Junda He, and David Lo. 2023. CCBERT: Self-Supervised Code Change Representation Learning. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 182–193.

[97] Xin Zhou, Ting Zhang, and David Lo. 2024. Large Language Model for Vulnerability Detection: Emerging Results and Future Directions. *ICSE NIER track* (2024).

[98] Zhou Zhou, Lili Bo, Xiaoxue Wu, Xiaobing Sun, Tao Zhang, Bin Li, Jiale Zhang, and Sicong Cao. 2022. SPVF: security property assisted vulnerability fixing via attention-based models. *Empirical Software Engineering* 27, 7 (2022), 171.

[99] Noah Ziems and Shaoen Wu. 2021. Security Vulnerability Detection Using Deep Learning Natural Language Processing. In *2021 IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2021, Vancouver, BC, Canada, May 10-13, 2021*. IEEE, 1–6. https://doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484500