GVI: Guided Vulnerability Imagination for Boosting Deep Vulnerability Detectors

Heng Yong[†§], Zhong Li[†‡*], Minxue Pan[†‡*], Tian Zhang[†§*], Jianhua Zhao[†§], Xuandong Li[†§]
[†]State Key Laboratory for Novel Software Technology, Nanjing University, China
[‡]Software Institute, Nanjing University, China
[§]School of Computer Science, Nanjing University, China
652023330041@smail.nju.edu.cn, {lizhong, mxp, ztluck, zhaojh, lxd}@nju.edu.cn

***Abstract*—The use of deep learning to achieve automated software vulnerability detection has been a longstanding interest within the software security community. These deep vulnerability detectors are mostly trained in a supervised manner, which heavily relies on large-scale, high-quality vulnerability datasets. However, the vulnerability datasets used to train deep vulnerability detectors frequently exhibit class imbalance due to the inherent nature of vulnerability data, where vulnerable cases are significantly rarer than non-vulnerable cases. This imbalance adversely affects the effectiveness of these detectors. A promising solution to address the class imbalance problem is to artificially generate vulnerable samples to enhance vulnerability datasets, yet existing vulnerability generation techniques are not satisfactory due to their inadequate representation of real-world vulnerabilities or their reliance on large-scale vulnerable samples for training the generation model.**

This paper proposes GVI, a novel approach aimed at generating vulnerable samples to boost deep vulnerability detectors. GVI takes inspiration from human learning with imagination and proposes exploring LLMs to imagine and create new, informative vulnerable samples from given seed vulnerabilities. Specifically, we design a Chain-of-Thought inspired prompt in GVI that instructs the LLMs to first analyze the seed to retrieve attributes related to vulnerabilities and then generate a set of vulnerabilities based on the seed's attributes. Our extensive experiments on three vulnerability datasets (i.e., Devign, ReVeal, and BigVul) and across three deep vulnerability detectors (i.e., Devign, ReVeal, and LineVul) demonstrate that the vulnerable samples generated by GVI are not only more accurate but also more effective in enhancing the performance of deep vulnerability detectors.

I. Introduction

Modern software is widely afflicted by security vulnerabilities [1], [2]. According to the NVD statistics [3], there have been 22,378 vulnerabilities publicly reported so far within 2023 alone. These vulnerabilities pose significant threats to the security of software: they often result in critical financial losses, legal consequences, and data breaches [4]. In response, considerable efforts have been dedicated to detecting these vulnerabilities, among which deep-learning-based approaches (i.e., deep vulnerability detectors) [5], [6], [7], [8], [9] have gained significant momentum in recent years due to their promise to overcome key limitations (e.g., imprecision and low coverage) of traditional code-analysis-based techniques [10], [11], [12].

Context. Despite the promise deep vulnerability detectors hold, the scarcity of high-quality vulnerability datasets presents a fundamental challenge to their advancement. Like any deep learning models, deep vulnerability detectors also heavily rely on quality and sizeable vulnerability datasets to be effective [13], [14]. Unfortunately, vulnerable code is relatively scarce because vulnerable cases are less likely to occur compared to normal cases in real-world software [15]. This leads to vulnerability datasets for deep vulnerability detectors typically suffering from class imbalance, with normal samples far outnumbering vulnerable ones. Deep vulnerability detectors trained with such imbalanced datasets tend to be skewed towards the normal class, resulting in poor generalization capability and sub-optimal performance [16]. Manually collecting more vulnerabilities to balance the normal and vulnerable samples is labor-intensive and challenging, making it clearly undesirable. Therefore, it is essential to address the class imbalance issue in vulnerability datasets to achieve more advanced deep vulnerability detectors.

State of the art. A number of approaches targeting the class imbalance issue have been proposed, such as re-sampling the dataset for a more balanced class distribution [17], [18], [19] or designing robust loss functions that account for class distribution [20], [21], which all share the commonality of acknowledging the dataset's imbalance and focusing on model training. Recently, a shift towards studying the dataset itself and involving the generation of more samples of minority classes (i.e., classes having few samples) through external knowledge to mitigate the class imbalance issue has emerged [22], [23]. In the context of vulnerability detection, researchers have explored automated vulnerability generation approaches for generating vulnerable samples to boost deep vulnerability detectors, with representatives such as VULGEN [24] and VGX [25]. In general, existing vulnerability generation approaches typically generate vulnerable samples through vulnerability injection, i.e., they inject vulnerabilities mined from a set of seed samples into normal programs to produce vulnerable samples.

Although the existing vulnerability generation approaches have been demonstrated to be quite conducive to boosting the performance of deep vulnerability detectors by augmenting the datasets with newly generated samples [24], [25], there is still room for improvement. Specifically, the existing vulnerability generation approaches involve mining vulnerability-injection patterns and learning localization models based on a seed set of vulnerable samples and their paired fixed versions. However, such datasets are often relatively small, as the pairs of vulnera-

* Corresponding authors.

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00214

ble samples and their corresponding repairs are also limited. As a result, the existing techniques learns ineffective localization models and vulnerability-injection patterns, leading to high noise in the generated samples. Furthermore, the vulnerability-injection patterns in existing approaches focus primarily on single-line edits, meaning the vulnerabilities injected involve changing only one line of code to be fixed. However, real-world vulnerabilities often involve multiple lines of code [26]. Hence, the vulnerable samples generated by existing work are less representative of real-world vulnerabilities, limiting the effectiveness of deep vulnerability detectors trained on them in detecting more complex, multi-line vulnerabilities.

Our approach. In this paper, we propose a simple yet effective method, called GVI, for generating vulnerable samples to enhance deep vulnerability detectors. The design idea of GVI is inspired by human learning with imagination [27], [28], i.e., humans can easily imagine different variants of a given object based on their prior understanding of the world. In light of this, GVI builds upon Large Language Models (LLMs) to simulate the imagination process for generating vulnerable samples. The insight of employing LLMs is that LLMs can implicitly acquire extensive knowledge about vulnerabilities because they are pre-trained on billions of code snippets, which can include numerous vulnerable programs; in addition, LLMs show strong capabilities in understanding, analyzing, and generating code [29]. By drawing on the knowledge and capabilities of LLMs, GVI is able to conduct imagination on a seed vulnerable code to create new vulnerable samples infused with new information.

Based on the above ideas, GVI incorporates a Chain-of-Thought (CoT) [30] inspired prompt to leverage LLMs for creating new vulnerable samples from a seed. Our insight is that conducting imagination in a single LLM call would result in generation with less guidance, thus leading to ambiguous vulnerable samples. In contrast, by adopting a CoT-inspired prompt, we make the LLMs imagine step-by-step, with each prompt building upon the previous one; in this way, more information is included in the imagination process, thereby enhancing the quality of the generated vulnerable samples. More specifically, GVI decomposes the imagination process into two phases: attribute sourcing, where the LLMs understand and analyze the seed to obtain attributes related to vulnerabilities, and vulnerability generation, where the LLMs generate a set of vulnerabilities based on the seed's attributes. As such, the attributes related to vulnerabilities in the seed inform the generation process, resulting in more accurate and effective vulnerable samples.

Results. To evaluate the effectiveness of GVI, we conduct an empirical study based on three vulnerability datasets, i.e., Devign [8], ReVeal [7], and Big-Vul [26], and three deep vulnerability detectors, i.e., Devign [8], ReVeal [7], and LineVul [9]. Our experimental results demonstrate that the vulnerable samples generated by GVI are not only more accurate (i.e., truly vulnerable) but also more effective in boosting deep vulnerability detectors. Specifically, the deep vulnerability detectors trained with GVI's generated samples consistently outperform those trained with other compared approaches across various settings. Furthermore, we demonstrate the contributions of our proposed CoT-inspired prompting strategy in GVI through ablation studies.

Summary. The main contributions of this paper are as follows:

- **Approach.** We propose GVI, a novel approach that instructs LLMs with a CoT-inspired strategy to generate vulnerable samples for enhancing deep vulnerability detectors.
- **Evaluation.** We extensively evaluate GVI using three vulnerability datasets and three deep vulnerability detectors. Experimental results demonstrate that the vulnerable samples generated by GVI are not only more accurate but also more effective in enhancing deep vulnerability detectors.
- **Artifact.** We have released our code as well as all the experimental data at: https://github.com/GVI24/GVI.

II. Background & Motivation

A. LLMs for Code Generation

LLMs have recently revolutionized the software engineering community, exhibiting remarkable prowess across a wide range of tasks [29], particularly in code generation, where they are used to automatically generate source code according to users' requirements [31]. Typically, an LLM, denoted as M, takes a prompt $\mathcal{P}$ (i.e., the user's requirement) as input and generates a sequence of tokens $\mathcal{C} = \{tk_0, tk_1, \ldots, tk_S\}$ which represents the expected code based on the prompt $\mathcal{P}$. More specifically, the generation process can be modeled as $p(\mathcal{C}) = \prod_{s=1}^{S} p(tk_s|tk_{0:s-1}, \mathcal{P})$, which calculates a distribution by decomposing the conditional probabilities of tokens, given their preceding context, enabling the prediction of the most probable token following a given sequence of tokens based on contexts. In this work, unlike prior research that focuses on leveraging LLMs to generate functionally correct code, we adopt LLMs to generate source code containing vulnerabilities to aid in the training of deep vulnerability detectors.

Prompt Engineering. Recent work demonstrates that the prompts provided to LLMs can significantly affect the performance of LLMs [32]. To better exploit LLMs, prompting engineering [33], which involves strategically optimize the prompts, has emerged as a crucial technique. Among the existing prompting engineering research, Chain-of-Thought (CoT) [30] shows enormous potential on a variety of common and complex tasks. Through a sequence of short sentences to describe reasoning logics step by step (known as reasoning chains or rationales), CoT could elicits the reasoning capabilities of LLMs, thus generating correct answers with a much greater chance. In this paper, we aim to explore CoT prompting strategies to guide LLMs in creating vulnerable samples for boosting the performance of deep vulnerability detectors.

B. Deep Vulnerability Detectors

A software vulnerability is a defect or a weakness in a software's design or implementation that can compromise its security, e.g., leading to system crashes or allowing attackers to gain control over the system [9]. To address these threats,

inspired by the success of deep learning, a rich line of deep-learning-based vulnerability detection approaches (denoted as Deep Vulnerability Detectors) has been developed recently [5], [6], [7], [8], [9]. Deep vulnerability detectors utilize DL models to learn patterns in code that indicate potential vulnerabilities. For example, Devign [8] employs a Gated Graph Neural Network (GGNN) [34] to automatically learn the properties of source code graphs, including Abstract Syntax Trees (AST), Control Flow Graphs (CFG), Data Flow Graphs (DFG), and code sequences. LineVul [9] uses a Transformer model to directly learn patterns from code sequences. In this work, instead of developing a new deep vulnerability detector, we focus on enhancing the learning process of deep vulnerability detectors by addressing a fundamental challenge, i.e., the class imbalance caused by the scarcity of vulnerabilities.

C. Motivation

Deep vulnerability detectors, like any deep learning models, heavily rely on high-quality and large-scale datasets of vulnerabilities to effectively train their DL models [15]. However, the vulnerable cases are less likely to happen than the normal cases in real-world scenarios, leading to an abundance of normal samples and a relative scarcity of vulnerable samples [15]. As a result, the vulnerability datasets typically suffer from the class imbalance issue, i.e., the number of normal samples is far more than that of vulnerable samples. This imbalance hinders the DL model's ability to learn about vulnerabilities effectively [13], [15], resulting in sub-optimal performance of the deep vulnerability detectors.

Existing Vulnerability Generation Approaches. One promising solution to address the class imbalance issue is to generate more vulnerable samples to balance the class distribution [22], [23]. Recently, there are approaches being proposed to automatically generate vulnerable samples to boost deep vulnerability detectors [22], [24], [25]. In general, the existing vulnerability generation approaches share two main steps: (1) learning the vulnerability-injection patterns, and (2) applying to a normal program the patterns that are compatible with it and thus making a vulnerable version of the program. For example, VulGen [24] first extracts vulnerability-injection patterns from pairs of vulnerable samples and their respective fixed versions. Then, it uses a Transformer-based localization model to pinpoint the exact statement in the code where the vulnerability-injection patterns can be applied. Lately, VGX [25] enhances VulGen by manually refining vulnerability-injection patterns and employing a more advanced localization model. Despite their impressive results, existing vulnerability generation approaches still have room to improve.

First, the vulnerable samples generated by existing approaches are subject to high noise. Existing vulnerability generation approaches typically learn localization models for identifying where to inject vulnerabilities and vulnerability-injection patterns for introducing vulnerabilities based on datasets of vulnerability fixes, which consist of vulnerable samples and their paired fixed versions. However, the available datasets of such vulnerability fixes are (even collectively) limited, restricting the effectiveness of the learned localization models and vulnerability-injection patterns, i.e., resulting in inaccurate locations for injecting vulnerabilities or false positive vulnerability-injection patterns. Furthermore, datasets of vulnerability fixes used in existing approaches typically constructed by static analyzers that originally designed for locating buggy code. Consequently, the discovered vulnerability-injection locations and patterns could be ambiguous, further deteriorating the effectiveness of the datasets for learning effective localization models and vulnerability-injection patterns. The latest technique, VGX [25], manually refines the vulnerability-injection patterns. However, manual curation of these patterns is tedious and clearly undesirable.

Second, the vulnerable samples generated by existing approaches are less representative of realistic vulnerabilities. Existing vulnerability generation approaches are designed to inject vulnerabilities though single-line code edits, i.e., they only change one line of code to inject vulnerabilities. However, real-world vulnerabilities often involve multiple lines of code. For example, in the BigVul dataset [26], one of the largest real-world vulnerability datasets, approximately 71% of vulnerabilities are multi-line. Therefore, the vulnerable samples generated by existing approaches are less representative of real-world scenarios, diminishing the effectiveness of deep vulnerability detectors trained on them in detecting more complex, multi-line vulnerabilities. Additionally, beyond the vulnerable statements, other parts of the vulnerable code can also vary. Simply modifying one line for vulnerability injection leaves the rest of the code untouched, limiting the diversity of generated vulnerable samples and further reducing the effectiveness of the learned deep vulnerability detectors.

III. Overview

A. Problem Formulation

We focus on deep vulnerability detectors that aim to detect whether a source file or application is potentially vulnerable [5], [8]. In particular, we consider detectors that analyze code at the function level, as this type of deep vulnerability detectors is the most widely studied in previous work [8], [7], [9]. As discussed earlier, vulnerability datasets typically suffer from class imbalance due to the scarcity of vulnerabilities, and existing vulnerability generation techniques still fall short in generating vulnerabilities to augment the datasets and mitigate the class imbalance issue. Therefore, a better effective approach is desired to generate effective vulnerabilities that can advance deep vulnerability detectors.

More specifically, the problem we want to address in this paper is: given a vulnerability dataset $\mathcal{D}_o = \{x_i, y_i\}_{i=1}^{n_o}$, where x_i denotes a function and y_i is the class label of x_i, indicating whether x_i is vulnerable or not;n_o represents the number of samples in $\mathcal{D}_o$. Our goal is to generate a set of new vulnerable samples $\mathcal{D}_s = \left\{x'_j, y'_j\right\}_{j=1}^{n_s}$ to enlarge the dataset $\mathcal{D}_o$, such that a deep vulnerability detector trained on the expanded dataset $\mathcal{D}_o \bigcup \mathcal{D}_s$ outperforms the model trained on the original dataset $\mathcal{D}_o$ significantly.

B. Overarching Idea

Our key idea to generate vulnerable samples is to leverage Large Language Models (LLMs) to produce more from the existing vulnerabilities in $\mathcal{D}_o$. This generation mechanism is inspired by human learning with imagination [27], [28]. When humans observe an object, they can readily imagine its different variants, such as code with various implementations, optimizations, or contexts, based on their accumulated prior knowledge. More specifically, through this imagination process, one can apply extensive prior knowledge to create code variants infused with new information from a seed vulnerable sample, thereby generating informative vulnerabilities beneficial for training deep vulnerability detectors. In tandem with this, recent LLMs have demonstrated exceptional abilities in code-related tasks [29]. We observe that LLMs, on the one hand, are pre-trained on billions of code snippets, which likely include numerous vulnerable programs. This pre-training enables them to implicitly acquire extensive knowledge about vulnerabilities, forming a foundation for vulnerability imagination. On the other hand, their profound code generation capabilities allow for high-quality code creation, facilitating the generation of realistic and valid vulnerable samples. Hence, LLMs naturally become our tool of choice for establishing our imagination pipeline for vulnerability generation.

To guide the imagination towards creating effective vulnerable samples for training deep vulnerability detectors, we explore a CoT-inspired prompting strategy for vulnerability imagination in GVI. Compared to naive prompts, which conduct imagination in a single LLM call (e.g.,"Please create n vulnerable functions as the given example"), CoT emphasizes a systematic and step-by-step reasoning process. With each prompt building upon the previous one, CoT facilitates the inclusion of additional information, thereby enhancing the quality of the final answers. Following this, we decompose the imagination process into two phases: attribute sourcing, where the LLMs understand and analyze the seed to obtain information about it, and vulnerability generation, where the LLMs generate a set of vulnerabilities based on the seed's attributes. More specifically, let x denote a seed code and M represent the LLM. GVI first prompts M with an attribute sourcing prompt $\mathcal{P}_{attr}$ to analyze x and extract attributes $\mathcal{A}$ related to vulnerabilities, i.e., $\mathcal{A} = M(x; \mathcal{P}_{attr})$. After that, GVI prompts M again with a generation prompt $\mathcal{P}_{gen}$, along with the attributes $\mathcal{A}$, to generate new vulnerabilities: $\mathcal{X}_{\mathsf{new}} = M(x; \mathcal{P}_{gen}, \mathcal{A})$. Such a CoT-inspired prompting strategy makes the imagination process more guided, resulting in more accurate and useful vulnerable samples for training deep vulnerability detectors.

Through leveraging LLMs to imagine vulnerable samples from seed data for vulnerability generation, GVI mitigates the limitations mentioned in §II-C by (1) utilizing the vast repository of knowledge within LLMs, thereby reducing the reliance on the vulnerability fixes, and (2) utilizing the generative capability of LLMs to generate vulnerable samples from scratch based on seed attributes, thereby enabling diverse appearances of generated vulnerable samples while ensuring their faithfulness. Moreover, it is important to note that we focus on exploring prompting strategies rather than fine-tuning strategies for LLMs to generate vulnerable samples because fine-tuning typically requires vast amounts of training data to ensure high-quality results, which may also suffer from the data dilemma observed in prior work [22], [24], [25].

IV. Design

Fig. 1 presents the overview of our proposed vulnerability generation approach, GVI. GVI takes a seed vulnerable sample as input and generates a set of new vulnerable samples based on this seed. The generation process consists of two phases: attribute scouring and vulnerability generation. In the attribute scouring phase, GVI leverages an LLM to analyze and understand the seed data, obtaining its attributes such as application scenario, vulnerability type, and vulnerability patterns. Then in the vulnerability generation phase, GVI employs the LLM again to generate a set of vulnerable samples based on the attributes of the seed data. Among the generation process, as discussed in §III-B, we use a CoT-inspired prompting strategy to make the LLM reason step-by-step. This strategy breaks the generation task into several sub-tasks, with each subsequent task building on the preceding ones, thus introducing more information into the LLM's reasoning process and leading to better generation results. Note that, following OpenAI's guidelines [35], we also use a system prompt to help the LLM understand task requirements and grasp the background knowledge of vulnerabilities. In addition, to mitigate low-quality or invalid samples, we incorporate a sample checking and filtering module in GVI to evaluate the vulnerable samples generated by the LLM. Once sufficient vulnerable samples are produced, one can combine these newly generated vulnerable samples with the original dataset and then train an advanced deep vulnerability detector. Next, we discuss details of individual phases.

A. Input

The input to GVI is a vulnerable sample that servers as a seed for LLMs to imagine new vulnerable samples. Prior work [36], [37] suggests that LLMs' generation results are limited in diversity and biased when made without any references. For instance, directly leveraging LLMs to imagine vulnerable samples based on a naive prompt like "Please generate a set of vulnerable functions" would result in repetitive, biased samples towards popular vulnerability types, and stylistically different from human-written ones. To address this, we employ a seed vulnerable sample to guide the imagination process. Using the seed vulnerable sample, LLMs are guided to generate code that references the seed, resulting in code that is stylistically similar to the seed and thus enhancing the quality of the generated samples. More importantly, by sampling different seed vulnerable samples, we can provide information on different vulnerabilities to LLMs, resulting in vulnerable samples with various vulnerability types. In particular, GVI uses a random sampling without replacement strategy to select

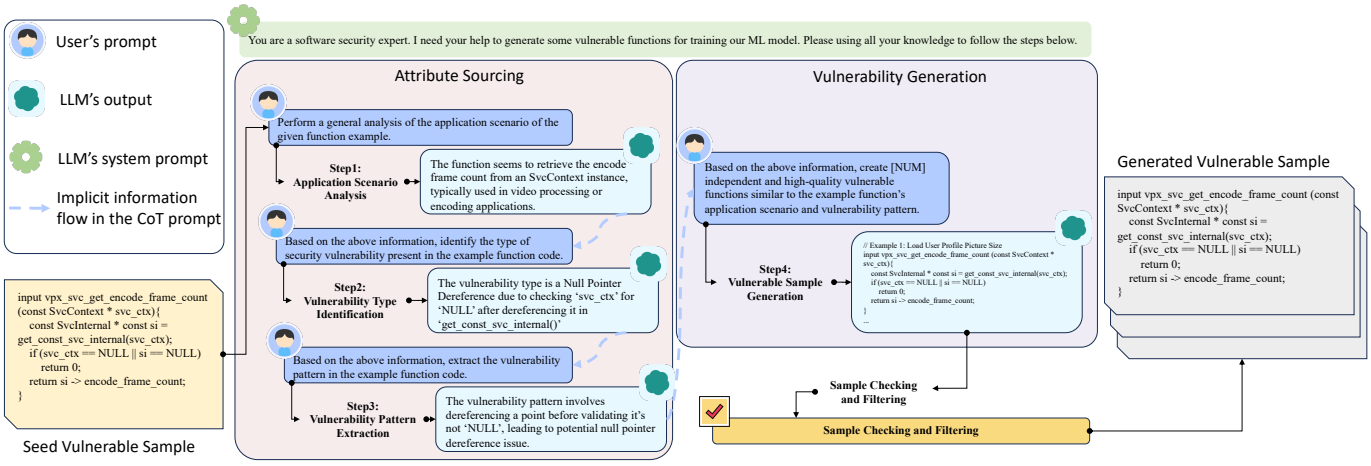

Fig. 1: Overview of GVI.

seed vulnerable samples from the original vulnerability dataset $\mathcal{D}_o$. Note that we mainly use the original vulnerability dataset as the seed pool to ensure that the generated samples are similar to those in the original dataset, thereby mitigating potential distribution shift threats to the deep vulnerability detectors.

B. Attribute Sourcing

In the attribute sourcing phase, GVI uses LLMs to extract attributes related to vulnerabilities from the seed data. To achieve this, we draw inspiration from the pipeline used by human security experts to discover vulnerabilities. According to [38], security experts typically use two main steps to identify vulnerabilities: first, they analyze the program to understand what the program does, and then they use their intuition and experience to find potential malicious actions. In light of this, we prompt LLMs to extract attributes related to vulnerabilities from the seed data in a step-by-step manner, as detailed below.

Step 1: Application Scenario Analysis. In alignment with human security experts, we first leverage LLMs to analyze the seed code to understand its behavior and context. To do this, we provide the seed code to the LLMs and ask the question:

> **Prompt:** *[Vulnerable Code]* Perform a general analysis of the application scenario of the given function example.

where the placeholder *[Vulnerable Code]* represents the seed vulnerable sample. As an example, the response of the LLM w.r.t the seed code shown in Fig. 1 is:

> **Response:** The function seems to retrieve the encode frame count from an SvcContext instance, typically used in video processing or encoding applications.

Step 2: Vulnerability Type Identification. Following the first step, we continue to identify the vulnerabilities behind the seed code. Specifically, we prompt the LLMs with the following prompt to summarize the vulnerability type of the seed. Note that the extensive prior knowledge of vulnerabilities in the LLMs, akin to the intuition and experience of human experts, provides a strong foundation for accurately identifying the vulnerability type.

> **Prompt:** Based on the above information, identify the type of security vulnerability present in the example function code.

As an example, the answer of the LLM w.r.t the seed code shown in Fig. 1 is:

> **Response:** The vulnerability type is a Null Pointer Dereference due to checking 'svc_ctx' for 'NULL' after dereferencing it in 'get_const_svc_internal()'

Step 3: Vulnerability Pattern Extraction. Beyond the vulnerability type, we further enrich the seed's vulnerability attributes with additional information on vulnerability patterns. This additional information provides more guidance in the subsequent vulnerability generation phase, resulting in more accurate vulnerable samples. In particular, the prompt used in GVI for extracting vulnerability patterns is:

> **Prompt:** Based on the above information, extract the vulnerability pattern in the example function code.

The example response w.r.t the seed code shown in Fig. 1 is as below:

> **Response:** The vulnerability pattern involves dereferencing a point before validating it's not 'NULL', leading to potential null pointer dereference issue.

C. Vulnerability Generation

After sourcing the attributes related to vulnerabilities from the seed data, GVI leverages LLMs to create vulnerable samples based on these attributes. Two key requirements for generated vulnerable samples that benefit training deep vulnerability detectors are faithfulness and diversity [39]. To achieve these objectives, we design the prompt to: (1) specify the desired properties of the outputs (i.e., following the extracted attributes), thus improving their quality, and (2) request the LLMs to create independent vulnerable samples from scratch instead of modifying the seed, thereby enabling more diverse outputs. In particular, the prompt used for generating vulnerable samples is:

Prompt: Based on the above information, create *[NUM]* independent and high-quality vulnerable functions similar to the example function's application scenario and vulnerability pattern.

where the placeholder *[NUM]* represents the required number of vulnerable samples generated. The example of vulnerable samples generated by LLMs are as follows:

Response:
// Example 1: Load User Profile Picture Size

```
input vpx_svc_get_encode_frame_count (
    const SvcContext * svc_ctx){
    const SvcInternal * const si =
        get_const_svc_internal(svc_ctx);
    if (svc_ctx == NULL || si == NULL)
        return 0;
    return si -> encode_frame_count;
}
...
```

As shown in the above example, the generated vulnerable samples have similar vulnerability attributes to the seed, sharing the same vulnerability type but exhibiting different implementations and semantics. These variations serve as good variants of the seed, which can effectively enhance the performance of deep vulnerability detectors [40].

Sample Checking and Filtering. Despite the strong generation capability of LLMs, the code they produce may also potentially suffer from syntax bugs (e.g., syntax invalid or syntax error) [41]. Vulnerable samples with syntax bugs are less useful for training deep vulnerability detectors, as they do not reflect real-world scenarios. Therefore, we further use a static analyzer to check the syntactical correctness of the generated samples and filter out those with errors.

V. Evaluation

We evaluate GVI on the following research questions:

- **RQ1:** How effectively can GVI generate vulnerable samples?
- **RQ2:** How effectively can the vulnerable samples generated by GVI boost deep vulnerability detectors?
- **RQ3:** How does the novel CoT-inspired prompting strategy in GVI contribute?

A. Experimental Settings

Vulnerability Datasets. We extensively evaluate the effectiveness of GVI on three vulnerability datasets: ReVeal [7], Devign [8], and Big-Vul [26]. We select these three datasets because they are built from real-world vulnerabilities in open-source projects and have been widely used in prior work on vulnerability detection [9], [7], [8], [26], [42]. In particular, Devign is collected from two large C projects, i.e., FFmpeg and QEMU, and consists of 14,858 normal samples and 12,460 vulnerable samples. ReVeal is collected from two large-scale open-source projects, i.e., Linux Debian Kernel and Chromium, and consists of 16,511 normal samples and 1,658 vulnerable samples. Big-Vul is larger in scale, constructed from the public Common Vulnerabilities and Exposures database, containing 177,736 normal samples and 10,900 vulnerable samples. Note that the samples in all three datasets are manually labeled as either normal or vulnerable at the function level.

TABLE I: Statistics of studied vulnerability datasets.

Dataset	# Samples			Imbalance Ratio
	# Normal	# Vulnerability	Total	
Devign	14,858	12,460	27,318	1.19
ReVeal	16,511	1,658	18,169	9.96
Big-Vul	177,736	10,900	188,636	16.31

Table I summarizes the statistical information of these three vulnerability datasets. In the table, Column "# Samples" presents the number of samples in each dataset, and Column "Imbalance Ratio" presents the ratio of normal samples to vulnerable samples. As shown, all three datasets suffer from a class imbalance issue, with the number of vulnerable samples being much smaller than the number of normal samples.

Deep Vulnerability Detectors. For deep vulnerability detectors, we consider Devign [8], ReVeal [7], and LineVul [9] as the subjects. These are the representative detectors built upon the Devign dataset, the Reval dataset, and the Big-Vul dataset, respectively. More specifically, Devign learns a GGNN [34] on a graph that combines the AST, CFG, DFG, and code sequence of the input code snippet for vulnerability detection. ReVeal combines Devign with resampling techniques [17] and the triplet loss [43] to detect vulnerabilities. LineVul detects vulnerabilities through sequence modeling of code as natural language tokens based on Transformer. Please note that in this work, we focus on exploring whether the vulnerable samples generated by GVI can enhance deep vulnerability detectors. Therefore, assessing whether the chosen detectors outperform all other options is not our primary concern. We believe that if GVI can improve the performance of the chosen detectors, it is likely to benefit other detectors as well.

Compared Approaches. In our study, we evaluate GVI by comparing it with the following three types of approaches.

Vulnerability Generation Approaches. We mainly compare GVI against the following two recent state-of-the-art vulnerability generation approaches.

- *VULGEN* [24] generates vulnerable samples by injecting vulnerabilities into normal code. It starts by mining vulnerability-injection patterns from a collection of vulnerable code and their corresponding fixed versions. Then, it fine-tunes a CodeT5 model [44] to locate where to apply these patterns, thereby creating new vulnerable samples.
- *VGX* [25] improves upon VULGEN in two key designs. First, it manually diversifies and refines the vulnerability-injection patterns. Second, it employs a more advanced semantics-aware contextualization model to better identify the locations for applying these patterns.

Imbalance Learning Approaches. Beyond the vulnerability generation approaches, we also consider several classical approaches that target the class imbalance issue to more comprehensively evaluate the effectiveness of GVI in enhancing

deep vulnerability detectors. Specifically, the following three approaches are considered in our experiments:

- *Oversampling* [45] refers to re-sampling techniques used to address the class imbalance issue. It balances the dataset by randomly selecting examples from the minority class (with replacement) and adding them to the training dataset.
- *FocalLoss* [46] stands for the re-weighting techniques for the class imbalance issue. It assigns higher weights to minority classes to balance their contribution to the overall loss function, thus achieving balanced model training.
- *SAM* [47] represents an advanced optimization technique for the class imbalance issue. It simultaneously minimizes loss and sharpness to enhance gradient components in negative curvature for effective saddle point escape, thereby improving generalization performance on minority classes.

Note that we exclude SMOTE [17], another classical approach targets the class imbalance issue, because the interpolation strategy of SMOTE is specific to numerical samples. However, the inputs of our subject deep vulnerability detectors are typically code or graphs, making SMOTE unsuitable.

LLMs for Vulnerability Detection. In addition, we evaluate the effectiveness of directly using LLMs for vulnerability detection to highlight the necessity of leveraging GVI for generating vulnerable samples to enhance deep vulnerability detectors. Specifically, we consider the following three typical and state-of-the-art prompting strategies for instructing LLMs.

- *Naive Prompting*. This directly asks the LLMs the simple question: "Is there a vulnerability in the given code?"
- *Few-shot Prompting*. This provides a few exemplars where example code and ground-truth answers are provided before the simple question in the prompt. In particular, we consider three settings in our experiments: 5-shot, 10-shot, and 20-shot where 5 exemplars, 10 exemplars, and 20 exemplars are provided, respectively.
- *CoT Prompting*. This improves upon naive prompting by incorporating reasoning steps. Specifically, we follow Kojima et al. [48] to achieve this by appending the statement "Let's think step by step" after the naive prompt.

Metrics. To evaluate the improvements GVI brings to deep vulnerability detectors, we follow prior work [24], [25] to measure the performance of deep vulnerability detectors using three widely-used metrics, i.e., Precision, Recall, and F1-Score. Let $\#TP$ denote the number of vulnerable samples correctly detected as vulnerabilities, $\#FP$ denote the number of normal samples incorrectly detected as vulnerabilities, and $\#FN$ denote the number of vulnerable samples incorrectly detected as normal. The three metrics are computed as: $Precision = \frac{\#TP}{\#TP+\#FP}$, $Recall = \frac{\#TP}{\#TP+\#FN}$, and $F1 = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$. Moreover, to investigate the statistical significance of the difference between two compared approaches, we apply the Wilcoxon rank-sum test [49] and the Cliff's delta δ effect size [50] to compare the performances of deep vulnerability detectors obtained by these two compared approaches. Based on the statistical testing results, we introduce "W/T/L" [51] to denote GVI performing better than, equal to, or worse than the baseline methods. In particular, we mark it as a "Win" if GVI performs significantly better according to the Wilcoxon rank-sum test value is less than 0.05 and meanwhile, the magnitude of the differences between the two methods is non-negligible according to Cliff's delta ($\delta \geq 0.147$). In contrast, we mark it as a "Lose"; Otherwise, we mark it as a "Tie".

Implementations. We implement GVI based on Python. By default, we configure GVI with OpenAI's GPT-4 model [35], which is currently one of the most effective LLMs due to its superior understanding and generative capabilities. It is important to note that GVI can be easily configured to work with other LLMs as well. For the parameters in GVI, we set the temperature of GPT-4 to 0.9 and request 4 samples from the LLM for each generation. To ensure our experiments within a computational friendly scale, we adjust the number of vulnerable samples generated based on the sizes and imbalance ratios of the subject vulnerability datasets. Specifically, we generate a total of $2,398$ vulnerable samples for Devign, $5,000$ for ReVeal, and $10,000$ for Big-Vul. Regarding the compared approaches, we directly use their open-source implementations and configure their parameters according to the settings recommended by the authors or the defaults specified in the original papers to ensure the accuracy of experimental repetition. When evaluating LLMs for vulnerability detection, we also use OpenAI's GPT-4 model [35] to ensure a fair comparison. For the subject deep vulnerability detectors, we also use their open-source implementations with default parameter settings. Note that our goal is to investigate whether the vulnerable samples generated by GVI can enhance deep vulnerability detectors, hence, tuning the parameters of the deep vulnerability detectors is beyond the scope of this work. Additionally, we perform 5 repetitions for each studied approach to mitigate the influence of randomness.

Experiment Environment. We deploy all experiments on a workstation with Intel Xeon W-2245 CPU, 64GB memory, and one NVIDIA GeForce RTX3090 GPU.

B. RQ1: Effectiveness in Generating Vulnerabilities

In this RQ, we first investigate the quality of vulnerable samples generated by GVI. To achieve this, we compare GVI with two vulnerability generation approaches, VULGEN [24] and VGX [25], by manually verifying whether the generated samples are indeed vulnerable. We follow prior work [24], [25] to design our user study. Specifically, we first randomly select $1,667$ examples from the samples generated by each approach. This sample size is statistically significant at the 99% confidence level and a 3% margin of error with respect to the population sizes of the generated vulnerable samples. Then, the first (Rater-1) and second (Rater-2) authors of this paper, along with a non-author PhD student (Rater-3), who each have over four years of experience in software engineering and security, manually review each sampled code. They label each code as vulnerable or not, following the vulnerability discovery process outlined by Votipka et al. [38]. In particular, the criterion for determining whether a sample is vulnerable is: a sample

TABLE II: Improvement of deep vulnerability detectors in terms of F1-Score.

Detector	Setting	Orig.	Vulnerability Generation		Imbalance Learning			GVI
			VULGEN	VGX	OverSampling	FocalLoss	SAM	
Devign	Training: Devign Testing: ReVeal	14.20	13.41 ↓	12.87 ↓	12.95 ↓	14.90 ↑	14.71 ↑	**16.65** ↑
	Training: Devign Testing: Big-Vul	8.21	8.31 ↑	7.41 ↓	7.74 ↓	8.56 ↑	7.93 ↓	**10.14** ↑
	Training: ReVeal Testing: Devign	12.40	42.57 ↑	44.75 ↑	34.16 ↑	48.32 ↑	11.44 ↓	**56.29** ↑
	Training: ReVeal Testing: Big-Vul	0.74	5.55 ↑	4.13 ↑	3.71 ↑	3.64 ↑	0.45 ↓	**11.10** ↑
	Training: Big-Vul Testing: Devign	0.02	40.77 ↑	52.03 ↑	2.94 ↑	42.12 ↑	0.01 ↓	**57.82** ↑
	Training: Big-Vul Testing: ReVeal	0.05	3.41 ↑	1.87 ↑	0.14 ↑	1.81 ↑	0.07 ↑	**13.67** ↑
ReVeal	Training: Devign Testing: ReVeal	14.43	15.29 ↑	15.00 ↑	15.12 ↑	15.28 ↑	15.14 ↑	**16.33** ↑
	Training: Devign Testing: Big-Vul	7.67	8.76 ↑	8.33 ↑	8.09 ↑	7.61 ↓	8.34 ↑	**10.07** ↑
	Training: ReVeal Testing: Devign	4.32	26.91 ↑	40.82 ↑	42.88 ↑	27.52 ↑	1.98 ↓	**53.52** ↑
	Training: ReVeal Testing: Big-Vul	0.66	7.22 ↑	2.89 ↑	7.18 ↑	5.54 ↑	2.51 ↑	**10.98** ↑
	Training: Big-Vul Testing: Devign	0.55	35.99 ↑	34.68 ↑	4.82 ↑	12.75 ↑	1.89 ↑	**48.49** ↑
	Training: Big-Vul Testing: ReVeal	0.24	10.21 ↑	9.58 ↑	0.69 ↑	0.26 ↑	0.12 ↓	**16.97** ↑
LineVul	Training: Devign Testing: ReVeal	12.63	15.57 ↑	14.95 ↑	14.85 ↑	14.79 ↑	13.81 ↑	**16.20** ↑
	Training: Devign Testing: Big-Vul	10.00	12.34 ↑	11.95 ↑	12.62 ↑	13.20 ↑	12.74 ↑	**14.16** ↑
	Training: ReVeal Testing: Devign	3.84	4.38 ↑	4.15 ↑	5.98 ↑	5.16 ↑	3.49 ↓	**62.36** ↑
	Training: ReVeal Testing: Big-Vul	2.16	2.55 ↑	1.82 ↓	6.76 ↑	6.30 ↑	2.26 ↑	**13.18** ↑
	Training: Big-Vul Testing: Devign	6.01	7.32 ↑	6.68 ↑	9.29 ↑	6.42 ↑	4.71 ↓	**19.17** ↑
	Training: Big-Vul Testing: ReVeal	5.98	8.82 ↑	8.02 ↑	8.63 ↑	5.02 ↓	4.46 ↓	**11.00** ↑

is labeled as vulnerable if a vulnerability can be triggered within the function. This aligns with the subject detectors that detect vulnerabilities at the function level. Please note that while $1,667$ samples may not be ideal, accurate automated vulnerability detection is not yet available.

Based on the manual labeling results, we first calculate the inter-rater agreements using Cohen's Kappa [49] to assess the quality of the manual labels. Specifically, the inter-rater agreements are 0.78 between Rater-1 and Rater-2, 0.81 between Rater-1 and Rater-3, and 0.80 between Rater-2 and Rater-3. These results suggest that the agreements between different raters are substantial, indicating the reasonable reliability of the manual labels for the samples [50]. For the samples generated by the different approaches, 90.6% of the samples from GVI are labeled as vulnerable, compared to 71.0% from VULGEN and 73.9% from VGX. The high proportion of vulnerable samples generated by GVI demonstrates its effectiveness in accurately producing vulnerable samples.

We also require the raters to assess the syntactical correctness of the samples during the manual review process. As a result, 74 out of $1,667$ samples generated by VULGEN contain syntax errors, 30 out of $1,667$ samples generated by VGX have syntax errors, while only 8 out of $1,667$ samples generated by GVI contains syntax errors. These observations suggest that GVI can produce more syntactically correct vulnerable samples compared to existing vulnerability generation approaches, which rely on hard-coded vulnerability injection methods.

Answer to RQ1. The vulnerable samples generated by GVI are more accurate (i.e., truly vulnerable) and have fewer syntax errors compared to those generated by the state-of-the-art vulnerability generation approaches.

C. RQ2: Effectiveness in Boosting Deep Vulnerability Detectors

This RQ aims to evaluate whether the vulnerable samples generated by GVI can boost the performance of deep vulnerability detectors. Similar to prior work [22], [24], [25], we adopt independent testing, where the training samples and testing samples, drawn from different datasets, are used to evaluate the performance of deep vulnerability detectors. More specifically, we use each of the three vulnerability datasets, i.e.,

Devign, ReVeal, and Big-Vul, as the training dataset for one of the three studied deep vulnerability detectors, Devign, ReVeal, and LineVul. The performance of each trained detector is then evaluated on the remaining two datasets. For GVI, as mentioned in §V-A, we generate a total of 2,398 vulnerable samples to augment Devign, 5,000 to augment ReVeal, and 10,000 to augment Big-Vul. For the two vulnerability generation approaches, VULGEN and VGX, we generate the same number of samples as GVI to augment the datasets for a fair comparison. Note that the datasets augmented with generated vulnerabilities are only used for training deep vulnerability detectors, while the original datasets are used for evaluating the detectors. For the three compared imbalance learning approaches, we directly apply them to the original training datasets to train the deep vulnerability detectors. When comparing GVI with the methods that use LLMs directly for vulnerability detection, considering the high cost of requesting GPT-4, we randomly sample $1,727$, $1,674$, and $1,826$ code snippets based on the class distribution to construct the testing dataset for the Devign dataset, the ReVeal dataset, and the Big-Vul dataset, respectively. All these sample sizes are statistically significant at the 99% confidence level and a 3% margin of error with respect to the population sizes of the subject datasets.

Table II reports the improvements in terms of F1-Score achieved by GVI, two vulnerability generation approaches, and three imbalance learning approaches. In the table, Column "Orig." represents the F1 scores obtained on the original datasets without adding generated vulnerabilities or applying imbalance learning approaches. Additionally, we use ↑to indicate improved performance compared to the 'Orig.', and ↓to indicate decreased performance compared to the 'Orig.'. Note that, as shown in the table, the models trained on ReVeal and Big-Vul typically have small F1 scores. This is reasonable since these two datasets are highly imbalanced (with imbalance ratios of 9.96 and 16.31, respectively), making it challenging to learn effective detectors on such imbalanced datasets. Table III presents the comparison results between GVI and the methods that use LLMs directly for vulnerability detection, in terms of F1-Score. In the table, Row "GVI" presents the average F1 scores achieved by deep vulnerability detectors trained with GVI on each testing dataset and detectors trained on a particular dataset are excluded from evaluation on the corresponding testing dataset (e.g., detectors trained on Devign are not evaluated on the Devign testing set). In both Table II and Table III, the best F1-Score is indicated in bold, and the second-best is underlined. Furthermore, entries are further highlighted with a gray background if GVI wins when compared with the baseline approach, according to p-values ($p < 0.05$) and Cliff's delta ($\delta \geq 0.147$). Please note that the results in terms of Precision and Recall follow a similar trend; these results are omitted here due to space limitations and are available in detail on our project homepage. From Table II and Table III, we have the following observations.

Comparison to Vulnerability Generation Approaches. We can observe from Table II that the deep vulnerability detectors trained based on GVI consistently achieve higher F1 scores compared to those trained based on VULGEN and VGX. For example, the F1 scores of LineVul trained with GVI are, on average, 324.33% and 379.64% higher than those achieved with VULGEN and VGX, respectively. The results of statistical testing further suggest that the improvements achieved by GVI are statistically significant, where all 36 comparison cases show a p-value less than 0.05 and a Cliff's delta larger than 0.147. These results demonstrate that GVI generates vulnerable samples, which enables the training of more effective deep vulnerability detectors compared to VULGEN and VGX. We also notice from Table II that both VULGEN and VGX fail to enhance deep vulnerability detectors in certain cases, whereas GVI achieves improvements across all cases. For example, for Devign trained on Devign and tested on ReVeal, the F1 scores of models trained with VULGEN and VGX decrease by 5.56% and 9.37%, respectively, compared to those trained on the original dataset. However, GVI achieves a 17.25% improvement in the same cases. These observations demonstrate the effectiveness of GVI-generated samples in enhancing deep vulnerability detectors. Moreover, we observe from Table II that the deep vulnerability detectors trained with VULGEN and VGX underperform compared to those trained using the imbalance learning approaches in some cases. For example, for LineVul trained on Big-Vul and tested on Devign, the detector trained with Oversampling achieves F1 scores that are 26.92% and 39.07% higher than those trained with VULGEN and VGX, respectively. In contrast, the F1 scores of the detector trained with GVI are 106.35% higher than those trained with Oversampling. These findings further confirm the effectiveness of GVI in enhancing deep vulnerability detectors.

TABLE III: Comparison of GVI with LLMs for vulnerability detection in terms of F1-score

Method	Testing: Devign	Testing: ReVeal	Testing: Big-Vul
Naive Prompting	<u>42.82</u>	11.28	6.49
5-shot Prompting	40.81	11.80	6.94
10-shot Prompting	41.04	12.31	7.77
20-shot Prompting	36.95	<u>12.41</u>	<u>8.54</u>
CoT Prompting	38.31	10.40	5.27
GVI	**49.61**	**15.14**	**11.61**

Comparison to Imbalance Learning Approaches. As shown in Table II, GVI consistently improves the performance of deep vulnerability detectors in all cases, while Oversampling, FocalLoss, and SAM fail to achieve improvements in 2, 2, and 9 out of 18 cases (3 detectors $\times$ 6 settings), respectively. Furthermore, the deep vulnerability detectors trained based on GVI consistently obtain significantly higher F1 scores compared to those trained with the three imbalance learning approaches in all 54 cases. These observations suggest that by adding newly generated vulnerabilities into the training datasets, GVI effectively mitigates the class imbalance issue behind the vulnerability datasets more effectively compared to existing imbalance learning approaches.

Comparison to Directly Applying LLMs. From Table III, we can see that deep vulnerability detectors trained with GVI outperform all prompting strategies of LLMs. For example, when evaluating on Big-Vul, the F1 scores of detectors trained with GVI are 78.89%, 67.29%, 49.42%, 35.95%, and 120.30%

higher than those of Naive Prompting, 5-shot Prompting, 10-shot Prompting, 20-shot Prompting and CoT Prompting, respectively. Regarding the results of the statistical testing, GVI demonstrates significant superiority over the prompting strategies of LLMs in all 15 cases. These results indicate that having more (labeled) vulnerable samples to train task-specific deep vulnerability detectors remains necessary despite the continued advancement of general-purpose LLMs. Furthermore, LLMs themselves would also become more effective for a task after being further fine-tuned on datasets specific to that task [51].

Answer to RQ2. GVI is more effective in enhancing deep vulnerability detectors compared to existing vulnerability generation and imbalance learning approaches. Furthermore, deep vulnerability detectors enhanced by GVI achieve significantly better detection performance than using general-purpose LLMs directly for vulnerability detection.

D. RQ3: Contributions of CoT-inspired Prompt

In this RQ, we conduct a series of ablation studies to further analyze the contribution of our proposed CoT-inspired prompting strategy for generating vulnerable samples. Specifically, we constructed ten variants of GVI:

1) **Naive.** We directly prompt the LLM with the instruction, "Please create n vulnerable functions", to generate vulnerable samples.
2) **w/o Attributes.** This omits the whole attribute scoring phase and directly request the LLM to generate vulnerable samples based on the seed using the prompt: "Please create n vulnerable functions similar to the given example."
3) **w/o Scenario.** This omits the application scenario analysis step from the attribute scoring phase.
4) **w/o Type.** This omits the vulnerability type identification step from the attribute scoring phase.
5) **w/o Pattern.** This omits the vulnerability pattern extraction step from the attribute scoring phase.
6) **Reordering the steps for attribute sourcing.** This fully permutes the steps of attribute sourcing (c.f., §IV-B), resulting in five variants of GVI.

Unless otherwise noted, we mainly perform our ablation studies using the Devign dataset as the original training dataset and the Devign model as the subject deep vulnerability detector. For each variant, we generate $2,398$ samples, matching the number of samples generated by GVI, to augment the training dataset. Then, the detectors are trained on the augmented datasets and evaluated on the ReVeal and Big-Vul dataset, respectively. The results on other datasets and detectors are similar and are therefore omitted.

Table IV summarizes the F1 scores of detectors trained based on vulnerable samples generated by each technique. In the table, Column "O:1324" indicates that GVI performs step 1 first, followed by step 3, then step 2, and finally step 4 when generating a vulnerable sample; and other columns in this table have similar interpretations. Also, we highlight the best model performance in bold and shade the background gray whenever GVI wins the compared approaches according to the statistical testing. From Table IV, we make the following observations. First, GVI significantly outperforms the naive variant by a remarkable margin, with improvements of 32.04% and 18.32% on ReVeal and Big-Vul respectively, indicating that providing LLMs with guidelines such as seed or vulnerability attributes is essential for generating effective vulnerable samples beneficial for training vulnerability detectors. Second, removing the attribute sourcing phase notably decreases the effectiveness of GVI, with models trained using GVI achieving on average 19.70% and 62.76% higher F1 scores than those trained without attributes, on ReVeal and Big-Vul respectively. This demonstrates that analyzing the seed code's attributes related to vulnerabilities is essential for effectively generating vulnerable samples. Third, GVI obtains significantly higher F1 scores than all its variants that remove or reorder the attribute sourcing steps. Specifically, the average improvements of GVI over these variants are 19.20% and 31.82% on ReVeal and Big-Vul, respectively. This indicates the effectiveness of GVI's attribute sourcing steps, which align with the pipeline commonly used by human security experts to discover vulnerabilities. Fourth, the application scenario analysis contributes more than other steps. The potential reason is that without understanding the scenario of the seed code, the identified vulnerability types and patterns may not be accurate, leading to noisy guidelines for the generation process.

TABLE IV: Ablation test for GVI.

Settings	Naive	w/o Attributes	w/o Scenario	w/o Type	w/o Pattern	GVI
Testing: ReVeal	12.61	13.91	14.19	14.81	14.35	**16.65**
Testing: Big-Vul	8.57	6.23	8.45	8.52	6.49	**10.14**
	O:1324	**O:2134**	**O:2314**	**O:3124**	**O:3214**	**GVI**
Testing: ReVeal	14.08	14.07	15.18	10.74	15.63	**16.65**
Testing: Big-Vul	7.43	6.68	8.60	8.41	7.65	**10.14**

TABLE V: Improvement of vulnerability localization models in terms of Top-10 accuracy.

Model	Orig.	VULGEN	VGX	OverSampling	FocalLoss	SAM	GVI
IVDetect	46.77	48.02 ↑	46.55 ↓	46.85 ↑	47.73 ↑	47.14 ↑	**50.22** ↑
LineVul	31.58	52.63 ↑	40.76 ↑	34.58 ↑	32.79 ↑	30.43 ↓	**63.76** ↑

Answer to RQ3. All steps in our proposed CoT-inspired prompting strategy contribute to the overall effectiveness of GVI, and maintaining their default order is essential for preserving the effectiveness of GVI.

VI. Discussion

More Vulnerability Analysis Tasks. Following previous work [22], [24], we mainly focus on deep vulnerability detectors in this study. While we have demonstrated the effectiveness of GVI in enhancing deep vulnerability detectors, an interesting question remains: Can the vulnerable samples generated by GVI improve other vulnerability-related models? To explore this, we examine two vulnerability localization models, IVDetect [42] and LineVul [9], enhanced by GVI. The training sets for both IVDetect and LineVul are the Big-Vul dataset [26], thus, we follow the same setting described in §V-A, augmenting the Big-Vul dataset with $10,000$ newly

generated vulnerable samples. Then, we employ the top-10 accuracy, which is commonly used in IVDetect [42] and LineVul [9], to evaluate the trained models on the ReVeal dataset [7]. The ReVeal dataset is adopted for testing because it provides the vulnerable lines for the $1,658$ vulnerable samples. Table V presents the comparison results. As shown in the table, GVI achieves the best overall performance among the studied techniques, obtaining Top-10 accuracies of 50.22% on IVDetect and 63.76% on LineVul. Specifically, GVI's average improvements are significantly higher than those of the comparison approaches across all 10 cases. These results indicate that the vulnerable samples generated by GVI are also useful for improving vulnerability localization models. In the future, we plan to further explore the effectiveness of GVI across more vulnerability analysis tasks.

Extensions of GVI. GVI has demonstrated remarkable effectiveness in generating vulnerable samples that boost deep vulnerability detectors. Nevertheless, we believe it can be further improved in the following aspects. First, GVI mainly grasps three types of attributes, i.e., application scenario, vulnerability type and vulnerability pattern, to guide vulnerability generation, following the pipeline used by human security experts to discover vulnerabilities. While there are other attributes that could be helpful for vulnerability generation (e.g., input-output behaviors), we plan to incorporate more attributes into GVI in the future. Second, GVI is currently configured with GPT-4, one of the most effective LLMs, but different LLMs can be easily supported. We will explore the effectiveness of GVI with various LLMs in our future work. Third, in this study, we focus on the function-level vulnerability detection, which is the primary focus of prior work [5], [6], [7], [8], [9]. Thus, we mainly use GVI to generate vulnerabilities at the function-level. However, since GVI imagines vulnerable samples based on the seed's attributes, we believe that GVI could also produce vulnerable code spanning multiple functions or files if provided with seeds that exhibit such attributes. For future work, we plan to explore the extension of GVI to generate vulnerabilities that span multiple functions or files.

Threats to Validity. The main threat to **internal** validity lies in the correctness of the implementations of GVI, the compared approaches, and experimental scripts. To reduce this, we adopt the open-source implementations of the compared approaches and build our approach GVI on state-of-the-art libraries, and carefully check the source code of GVI and the experimental scripts. Another threat to **internal** validity lies in the manual analysis used to evaluate the correctness of the generated vulnerabilities. To mitigate this, we employ three independent raters and assess the inter-rater agreement using Cohen's Kappa [49]. The main threat to **external** validity lies in the selection of studied vulnerability datasets and deep vulnerability detectors. To mitigate this, we conduct experiments using three datasets derived from real-world vulnerabilities and are widely used in prior research, and we select three representative detectors built upon these datasets as our subjects. The main threat to **construct** validity lies in the metrics used in our study. To reduce this, we consider in total three metrics, namely Precision, Recall, and F1-Score, which have been widely used in related work [24], [25].

VII. Related Work

Deep Vulnerability Detectors. With the increased popularity of deep learning, many deep learning-based vulnerability detectors have been developed in recent years [5], [6], [7], [8], [9]. Despite these advancements, existing deep vulnerability detectors often exhibit a significant gap between their reported performance and their accuracy in detecting unseen, real-world vulnerabilities due to limitations in the size and quality of their training datasets [14]. One solution to this issue is addressing class imbalance through minority oversampling [7], while a recent study has highlighted that augmenting training data with more vulnerable samples is another important direction [22]. Our study explores this direction and proposes a novel and effective vulnerability generation method to enhance deep vulnerability detectors.

Vulnerability Generation. Besides the compared vulnerability generation approaches in our experiments (i.e., VULGEN [24] and VGX [25]), there are some other vulnerability generation approaches in the literature [52], [53], [54], [55]. For example, FixReverter [52] uses known bug-fix patterns in reverse to inject vulnerabilities. In comparison, GVI leverages LLMs with a CoT-inspired prompting strategy to generate vulnerable samples with the goal of enhancing deep vulnerability detectors.

Imbalance Learning. A number of imbalance learning approaches to address the class imbalance issue in training datasets have been developed recently, such as re-sampling [17], [18], [19] and class-balanced loss [20], [21], which all share the commonality of acknowledging the dataset's imbalance and focusing on model training. Unlike these approaches, GVI shifts focus towards the dataset itself and addresses the class imbalance issue by leveraging LLMs to generate additional vulnerable samples to balance the datasets.

VIII. Conclusion

In this paper, we propose GVI, a novel vulnerability generation approach aimed at boosting deep vulnerability detectors. GVI takes inspiration from human learning with imagination and proposes exploring LLMs to imagine and create new, informative vulnerable samples from given seed vulnerabilities. Specifically, GVI adopts a Chain-of-Thought prompt that instructs the LLMs to first analyze the seed to retrieve vulnerability attributes and then generate a set of vulnerabilities based on these attributes. Experimental results on three vulnerability datasets and across three deep vulnerability detectors demonstrate that the vulnerable samples generated by GVI are not only more accurate but also more effective in enhancing the performance of deep vulnerability detectors.

Acknowledgment

We thank the anonymous reviewers for their valuable feedback. This research was supported by the National Natural Science Foundation of China under Grant Nos. 62402214, 62372227 and 62232014, and the Natural Science Foundation of Jiangsu Province under Grant No. BK20241194.

REFERENCES

[1] X. Fu and H. Cai, "A dynamic taint analyzer for distributed systems," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, M. Dumas, D. Pfahl, S. Apel, and A. Russo, Eds. ACM, 2019, pp. 1115–1119. [Online]. Available: https://doi.org/10.1145/3338906.3341179

[2] W. Li, J. Ming, X. Luo, and H. Cai, "Polycruise: A cross-language dynamic information flow analysis," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 2513–2530. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/li-wen

[3] I. T. L. at NIST, "National vulnerability database (nvd) dashboard," [Online], https://nvd.nist.gov/general/nvd-dashboard.

[4] F. T. Council, "Zero-day vulnerabilities: 17 consequences and complications." [Online], https://www.forbes.com/sites/forbestechcouncil/2023/05/26/zero-day-vulnerabilities17-consequences-and-complications/?sh=711e37204b41.

[5] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_03A-2_Li_paper.pdf

[6] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 4, pp. 2244–2258, 2022. [Online]. Available: https://doi.org/10.1109/TDSC.2021.3051525

[7] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet?" *IEEE Trans. Software Eng.*, vol. 48, no. 9, pp. 3280–3296, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2021.3087402

[8] Y. Zhou, S. Liu, J. K. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 10 197–10 207. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html

[9] M. Fu and C. Tantithamthavorn, "Linevul: A transformer-based line-level vulnerability prediction," in *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*. ACM, 2022, pp. 608–620. [Online]. Available: https://doi.org/10.1145/3524842.3528452

[10] A. Austin, C. Holmgreen, and L. A. Williams, "A comparison of the efficiency and effectiveness of vulnerability discovery techniques," *Inf. Softw. Technol.*, vol. 55, no. 7, pp. 1279–1288, 2013. [Online]. Available: https://doi.org/10.1016/j.infsof.2012.11.007

[11] P. Li and B. Cui, "A comparative study on software vulnerability static analysis techniques and tools," in *2010 IEEE International Conference on Information Theory and Information Security*, Dec 2010. [Online]. Available: http://dx.doi.org/10.1109/icitis.2010.5689543

[12] Y. Nong, H. Cai, P. Ye, L. Li, and F. Chen, "Evaluating and comparing memory error vulnerability detectors," *Inf. Softw. Technol.*, vol. 137, p. 106614, 2021. [Online]. Available: https://doi.org/10.1016/j.infsof.2021.106614

[13] R. Croft, M. A. Babar, and M. M. Kholoosi, "Data quality for software vulnerability datasets," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 121–133. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00022

[14] Y. Nong, R. Sharma, A. Hamou-Lhadj, X. Luo, and H. Cai, "Open science in software engineering: A study on deep learning-based vulnerability detection," *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 1983–2005, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2022.3207149

[15] Z. Li, M. Pan, Y. Pei, T. Zhang, L. Wang, and X. Li, "Robust learning of deep predictive models from noisy and imbalanced software engineering datasets," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 86:1–86:13. [Online]. Available: https://doi.org/10.1145/3551349.3556941

[16] Y. Zhang, B. Kang, B. Hooi, S. Yan, and J. Feng, "Deep long-tailed learning: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 9, pp. 10 795–10 816, 2023. [Online]. Available: https://doi.org/10.1109/TPAMI.2023.3268118

[17] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002. [Online]. Available: https://doi.org/10.1613/jair.953

[18] H. Han, W. Wang, and B. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing, International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I*, ser. Lecture Notes in Computer Science, D. Huang, X. S. Zhang, and G. Huang, Eds., vol. 3644. Springer, 2005, pp. 878–887. [Online]. Available: https://doi.org/10.1007/11538059_91

[19] L. Shen, Z. Lin, and Q. Huang, "Relay backpropagation for effective learning of deep convolutional neural networks," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9911. Springer, 2016, pp. 467–482. [Online]. Available: https://doi.org/10.1007/978-3-319-46478-7_29

[20] K. Cao, C. Wei, A. Gaidon, N. Aréchiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 1565–1576. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/621461af90cadfdaf0e8d4cc25129f91-Abstract.html

[21] Y. Cui, M. Jia, T. Lin, Y. Song, and S. J. Belongie, "Class-balanced loss based on effective number of samples," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 9268–9277. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Cui_Class-Balanced_Loss_Based_on_Effective_Number_of_Samples_CVPR_2019_paper.html

[22] Y. Nong, Y. Ou, M. Pradel, F. Chen, and H. Cai, "Generating realistic vulnerabilities via neural code editing: an empirical study," 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*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1097–1109. [Online]. Available: https://doi.org/10.1145/3540250.3549128

[23] J. Shao, K. Zhu, H. Zhang, and J. Wu, "Diffult: How to make diffusion model useful for long-tail recognition," *CoRR*, vol. abs/2403.05170, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.05170

[24] Y. Nong, Y. Ou, M. Pradel, F. Chen, and H. Cai, "VULGEN: realistic vulnerability generation via pattern mining and deep learning," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 2527–2539. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00211

[25] Y. Nong, R. Fang, G. Yi, K. Zhao, X. Luo, F. Chen, and H. Cai, "VGX: large-scale sample generation for boosting learning-based software vulnerability analyses," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 149:1–149:13. [Online]. Available: https://doi.org/10.1145/3597503.3639116

[26] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A C/C++ code vulnerability dataset with code changes and CVE summaries," in *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*, S. Kim, G. Gousios, S. Nadi, and J. Hejderup, Eds. ACM, 2020, pp. 508–512. [Online]. Available: https://doi.org/10.1145/3379597.3387501

[27] A. Vyshedskiy, "Neuroscience of imagination and implications for human evolution," 2019. [Online]. Available: https://doi.org/10.31234/osf.io/skxwc

[28] J.-P. Sartre, *The psychology of the imagination*. Routledge, 2013.

[29] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. C. Grundy, and H. Wang, "Large language models for software

engineering: A systematic literature review," *CoRR*, vol. abs/2308.10620, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.10620

[30] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[31] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *CoRR*, vol. abs/2406.00515, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2406.00515

[32] D. Noever, "Can large language models find and fix vulnerable software?" *CoRR*, vol. abs/2308.10345, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.10345

[33] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language models: Techniques and applications," *CoRR*, vol. abs/2402.07927, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.07927

[34] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.05493

[35] OpenAI, "Gpt-4 documentation," [Online], https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo.

[36] Y. Yu, Y. Zhuang, J. Zhang, Y. Meng, A. J. Ratner, R. Krishna, J. Shen, and C. Zhang, "Large language model as attributed training data generator: A tale of diversity and bias," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/ae9500c4f5607caf2eff033c67daa9d7-Abstract-Datasets_and_Benchmarks.html

[37] A. Divekar and G. Durrett, "Synthesizrr: Generating diverse datasets with retrieval augmentation," *CoRR*, vol. abs/2405.10040, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.10040

[38] D. Votipka, R. Stevens, E. M. Redmiles, J. Hu, and M. L. Mazurek, "Hackers vs. testers: A comparison of software vulnerability discovery processes," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 374–391. [Online]. Available: https://doi.org/10.1109/SP.2018.00003

[39] L. Long, R. Wang, R. Xiao, J. Zhao, X. Ding, G. Chen, and H. Wang, "On llms-driven synthetic data generation, curation, and evaluation: A survey," *CoRR*, vol. abs/2406.15126, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2406.15126

[40] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *J. Big Data*, vol. 6, p. 60, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[41] S. Dou, H. Jia, S. Wu, H. Zheng, W. Zhou, M. Wu, M. Chai, J. Fan, C. Huang, Y. Tao *et al.*, "What's wrong with your code generated by large language models? an extensive study," *arXiv preprint arXiv:2407.06153*, 2024.

[42] Y. Li, S. Wang, and T. N. Nguyen, "Vulnerability detection with fine-grained interpretations," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 292–303. [Online]. Available: https://doi.org/10.1145/3468264.3468597

[43] C. Mao, Z. Zhong, J. Yang, C. Vondrick, and B. Ray, "Metric learning for adversarial robustness," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 478–489. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract.html

[44] Y. Wang, W. Wang, S. R. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds. Association for Computational Linguistics, 2021, pp. 8696–8708. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.685

[45] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009. [Online]. Available: https://doi.org/10.1109/TKDE.2008.239

[46] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 2999–3007. [Online]. Available: https://doi.org/10.1109/ICCV.2017.324

[47] H. Rangwani, S. K. Aithal, M. Mishra, and V. B. R., "Escaping saddle points for effective generalization on class-imbalanced data," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/8f4d70db9ecec97b6723a86f1cd9cb4b-Abstract-Conference.html

[48] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html

[49] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[50] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data." *Biometrics*, vol. 33 1, pp. 159–74, 1977. [Online]. Available: https://api.semanticscholar.org/CorpusID:11077516

[51] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, H. Zheng, J. Chen, Y. Liu, J. Tang, J. Li, and M. Sun, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nat. Mac. Intell.*, vol. 5, no. 3, pp. 220–235, 2023. [Online]. Available: https://doi.org/10.1038/s42256-023-00626-4

[52] Z. Zhang, Z. Patterson, M. Hicks, and S. Wei, "FIXREVERTER: A realistic bug injection methodology for benchmarking fuzz testing," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 3699–3715. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/zhang-zenong

[53] J. Patra and M. Pradel, "Semantic bug seeding: a learning-based approach for creating realistic bugs," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 906–918. [Online]. Available: https://doi.org/10.1145/3468264.3468623

[54] S. Roy, A. Pandey, B. Dolan-Gavitt, and Y. Hu, "Bug synthesis: challenging bug-finding tools with deep faults," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 2018, pp. 224–234. [Online]. Available: https://doi.org/10.1145/3236024.3236084

[55] H. Lee, S. Kim, and S. K. Cha, "Fuzzle: Making a puzzle for fuzzers," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 45:1–45:12. [Online]. Available: https://doi.org/10.1145/3551349.3556908