Vul-R2: A Reasoning LLM for Automated Vulnerability Repair

Xin-Cheng Wen[1†], Zirui Lin[2], Yijun Yang[1], Cuiyun Gao[3*], Deheng Ye[1‡]
[1] Tencent Inc., Shenzhen, China
[2] Department of Computer Science, City University of Hong Kong, China
[3] Department of Computer Science and Engineering, The Chinese University of Hong Kong, China
xiamenwxc@foxmail.com, ryanlzr2003@gmail.com, yijun.steven.yang@gmail.com,
cuiyungao@outlook.com, dericye@tencent.com

***Abstract*—The exponential increase in software vulnerabilities has created an urgent need for automatic vulnerability repair (AVR) solutions. Recent research has formulated AVR as a sequence generation problem and has leveraged large language models (LLMs) to address this problem. Typically, these approaches prompt or fine-tune LLMs to generate repairs for vulnerabilities directly. Although these methods show state-of-the-art performance, they face the following challenges: (1) Lack of high-quality, vulnerability-related reasoning data. Current approaches primarily rely on foundation models that mainly encode general programming knowledge. Without vulnerability-related reasoning data, they tend to fail to capture the diverse vulnerability repair patterns. (2) Hard to verify the intermediate vulnerability repair process during LLM training. Existing reinforcement learning methods often leverage intermediate execution feedback from the environment (e.g., sandbox-based execution results) to guide reinforcement learning training. In contrast, the vulnerability repair process generally lacks such intermediate, verifiable feedback, which poses additional challenges for model training.**

To address these challenges, we propose to model the vulnerability repair task from a reasoning perspective and train a reasoning LLM termed *Vulnerability Reasoner and Repair* (Vul-R2) which consists of two key modules: (1) a domain-aware reasoning learning module, which comprises a reasoning answer construction component, a reasoning data filtering process, and a supervised fine-tuning process for learning vulnerability-related reasoning knowledge; and (2) a curriculum-based verifiable rewarded training module, which comprises dynamically reinforcement learning with verifiable rewards paradigms based on multiple-choice question answering in an easy stage and character-level matching in a hard stage. We evaluate Vul-R2 on the real-world C/C++ dataset PrimeVul to demonstrate its effectiveness in vulnerability repair. Specifically, Vul-R2 outperforms the best baseline by 11.27% for exact match (EM) and successfully repairs 49 additional vulnerabilities. Furthermore, we demonstrate the effectiveness of the proposed paradigm, fine-tuning Vul-R2 on PrimeVul leads to improved EM performance of 8.78% on a human curated dataset SVEN, even without additional training.

***Index Terms*—Vulnerability Repair, Large Language Model, Reinforcement Learning**

† Work done at Tencent Inc.
* Corresponding author.
‡ Project leader.

I. Introduction

The explosive complexity and scale of contemporary software systems have led to a significant increase in software vulnerabilities [1]. Recent advancements in vulnerability detection [2] have further facilitated the identification and reporting of these vulnerabilities. According to the 2024 Synopsys report [3], 84% of codebases contain at least one open-source vulnerability. Unaddressed vulnerabilities pose substantial risks, including financial loss, data breaches, and systemic failures [4]–[7]. For instance, Cybersecurity Ventures projects that the global cost of cyberattacks will reach $9.5 trillion in 2024, with ransomware, phishing, and data breaches constituting the primary drivers of this increase [8]. Despite the critical importance of timely vulnerability remediation, the repair process remains highly labor-intensive [9]. Security experts must conduct root cause analyses, validate proposed fixes, and ensure compatibility across the codebase, often requiring hundreds of developer hours to address a single critical flaw. Recent years have witnessed the emergence of numerous automated vulnerability repair (AVR) methods, which aim to automate the vulnerability remediation process and reduce the manual effort required from developers.

The earliest proposed AVR techniques are rule-based methods [10]–[13]. It predominantly employs static [14], [15] or dynamic analysis [16] grounded in predefined rule sets. Although effective within their prescribed domains, the performance of these approaches is constrained by a limited coverage of vulnerability types, hindering their generalization to the diverse and continuously evolving landscape of software vulnerabilities in practice.

Inspired by the remarkable achievements of deep learning in the natural language processing (NLP) field [17]–[22], code pre-trained models (CodePTMs) (i.e., CodeT5 [23]) and large language models (LLMs) (i.e., ChatGPT [24]) are widely adopted for AVR. CodePTM-based methods [1], [25], [26] take the vulnerable code as input and generate the corresponding repaired code as output to address software vulnerabilities. They can naturally avoid the low coverage problem of rule-based approaches, as CodePTMs are rule-agnostic and make

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00011

predictions based on probability [27]. Despite the effectiveness, CodePTM-based AVR approaches perform badly on rare types and are overly concerned with semantic information. The recent emergence and rapid advancement of LLMs have substantially elevated the performance of automated code analysis and repair systems [28], rendering them highly effective for AVR. LLM-based methods [29] have achieved improvements in repair accuracy and coverage. These approaches benefit from pre-training on large-scale code datasets [30], pushing a rapid paradigm shift in the realm of vulnerability repair. Despite the impressive success, they still face challenges in learning the vulnerability repair patterns:

(1) Lack of high-quality, vulnerability-related reasoning data. While recent efforts have sought to enhance LLMs through techniques such as retrieving similar code snippets via chain-of-thought (CoT) [31] prompting or supervised fine-tuning (SFT) [32], these approaches primarily rely on foundation models that mainly encode general programming knowledge. Due to the complex vulnerability trigger patterns, the absence of explicit reasoning data tends to limit the model's ability to capture the diverse vulnerability repair strategies. For example, as shown in Fig. 1, QwQ fails to capture the underlying root cause of vulnerability. Specifically, since `pixel_value` is a 64-bit variable, the data type of the shift operand should also be 64-bit to prevent integer overflow. However, QwQ incorrectly treated it as 32-bit during its reasoning process. **(2) Hard to verify the intermediate vulnerability repair process during LLM training.** Effective vulnerability repair necessitates a multi-step planning process about value ranges and temporal relationships among symbolic variables [33]. However, existing reinforcement learning-based approaches often leverage intermediate execution feedback from the environment (e.g., sandbox-based execution results) to guide training. The vulnerability repair process generally lacks such intermediate, verifiable feedback, which poses challenges for model learning. As illustrated in Fig. 1, with the incorrect reasoning process, QWQ [34] produces a wrong result, which could be alleviated by intermediate verification.

Our work. To address the above challenges, we propose modeling the vulnerability repair task from a reasoning perspective, rather than relying on traditional prompt-based or SFT paradigms. Specifically, we formulate vulnerability repair as an exploratory-feedback problem, where the objective is to employ step-by-step reasoning to identify correct fixes from a large set of candidate solutions generated by LLMs, with the process being guided by verifiable feedback. It closely mirrors the way humans learn and reason through interactive feedback.

Specifically, we propose a reasoning LLM for vulnerability repair named *Vul-R2*, which consists of two key components: (1) A domain-aware reasoning learning module, which comprises a reasoning answer construction step for generating vulnerability-related reasoning data, a data filtering process to mitigate the impact of low-quality data, and an SFT process to bootstrap the model's understanding of vulnerability-related concepts. (2) A curriculum-based verifiable rewarded training module, enabling the model to progressively learn reasoning capabilities from an easy-to-hard stage. In the easy stage, we design multiple-choice questions by dynamically constructing verifiable reward signals. In the hard stage, we utilize character-level matching to further enhance the model's repair capabilities through reinforcement learning with verifiable rewards (RLVR).

To evaluate the effectiveness of Vul-R2, we compare it with seven existing vulnerability repair baselines on the two high-quality C/C++ benchmark datasets: PrimeVul [35] and SVEN [36]. Experimental results show that Vul-R2 outperforms the best baseline by 11.27% for exact match (EM) and successfully repairs 49 additional vulnerabilities in PrimeVul. Furthermore, we validate the generalization capability of Vul-R2 across another dataset, training a model on PrimeVul can help improve performance on SVEN.

Contributions. The major contributions of this paper are summarized as follows:

1) To the best of our knowledge, this is the first work exploring reasoning LLM for the vulnerability repair.
2) We propose Vul-R2, a reasoning LLM for vulnerability repair. Vul-R2 effectively enables reasoning about complex vulnerability patterns through the domain-aware reasoning and curriculum-based verifiable rewarded training.
3) We extend the PrimeVul dataset with CoT reasoning answers for vulnerability repair and conduct an extensive evaluation. The results demonstrate the effectiveness of Vul-R2 compared with baseline AVR methods.

II. Proposed Framework

A. Overview

Fig. 2 presents an overview of the proposed Vul-R2. The primary objective of Vul-R2 is to employ a reasoning process, enabling the model to first acquire fundamental reasoning skills. Guided by reinforcement learning with verifiable feedback, the model then iteratively selects the optimal reasoning steps to accurately repair vulnerabilities, progressing from easy to hard stages.

First, the process begins with the domain-aware reasoning learning module, which serves as a cold-start phase to familiarize the model with the reasoning process. This module comprises three steps: the reasoning answer construction, which generates the vulnerability-related reasoning data; the data filtering, which mitigates the impact of low-quality data; and an SFT training objective, which integrates domain-specific knowledge into the model. **Subsequently**, Vul-R2 employs a curriculum-based verifiable rewarded training module. It consists of two stages, starting with multiple-choice questions in the easy stage and then advancing to complex vulnerability repair tasks in the hard stage.

B. Domain-Aware Reasoning Learning (DARL)

We propose the domain-aware reasoning learning module for facilitating the initial learning of vulnerability-specific reasoning knowledge. As shown in Fig. 2 (A), it mainly contains three steps, including (a) reasoning answer construction, (b)

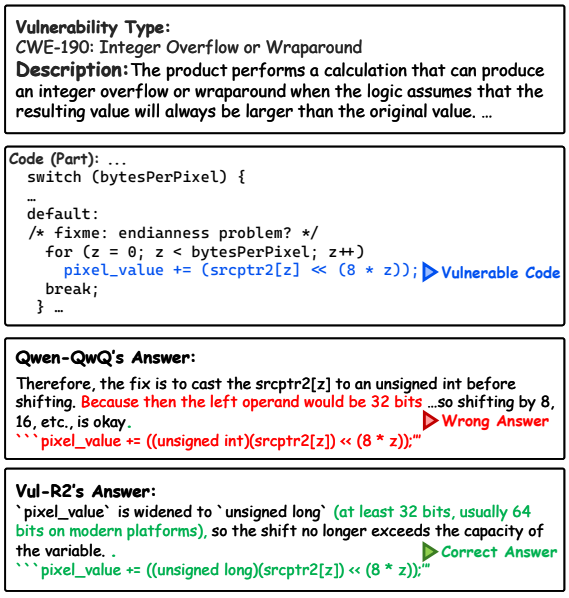


Fig. 1: Illustration of the vulnerability "Integer Overflow or Wraparound [37]" repaired by an open-source reasoning LLM (i.e., QwQ-32B [34]) and our Vul-R2. More detailed reasoning traces and the case of Vul-R2 can be found in Fig. 8. We adopt QwQ-32B in the case since the reasoning process of privileged LLMs, such as OpenAI-o3 [38], is inaccessible due to their usage policies.

reasoning data filter, and (c) domain-aware SFT training, with details as below.

a) Reasoning Answer Construction (RAC): This component aims at generating the vulnerability-related reasoning data. In real-world scenarios, developers' reasoning processes for identifying the root causes of vulnerabilities involve more than simply concluding directly. To this end, we propose a reasoning prompt to generate the solution steps c for vulnerability repair. Specifically, we construct the reasoning prompt as illustrated in Fig.3 (1)-(3). The prompt comprises three components: (1) instructions incorporating vulnerability domain knowledge, (2) repair-step guidance, and (3) reasoning specifications coupled with inverse verification.

(1) Instructions with domain knowledge: It clarifies the task requirements and outlines the steps to be executed by the LLMs. We provide explicit reasoning instructions, encouraging the LLM to first internally deliberate on the reasoning process before presenting the final answer to the user. Additionally, we incorporate common weakness enumeration (CWE) [39] vulnerability information and detailed vulnerability descriptions.

(2) Repair-step guidance: We summarize three key steps to reason the root cause of vulnerability in the target sample: analyzing the buggy code, performing trigger tests, and examining error messages. Specifically, the model initially performs a thorough analysis of the annotated vulnerable code; subsequently, it applies trigger tests, such as boundary condition assessments; finally, it corroborates whether the observed outcomes correspond with the reported error messages.

(3) Reasoning specifications with inverse verification: It aims to standardize the output format. We provide reference answers to enable the model to perform a secondary verification of its own responses. LLMs are instructed to follow the given format step-by-step and produce results accordingly. Based on the reference answers, the model may overturn its previous conclusions and generate a more precise reasoning process. The verification step helps further mitigate hallucinations and improve the reliability of the model's outputs.

b) Reasoning Data Filter: To ensure the quality of the constructed reasoning data, we introduce the data filter process to prevent misleading the model during training. As shown in Fig. 2, the data filter process mainly includes (a) model-based filtering and (b) rule-based filtering steps.

Model-based filtering. Inspired by prior work [40], the model-based filtering treats an LLM as a judge. In this process, the LLM functions as a binary classifier, responding with "yes" or "no". We retained those samples for which the model's judgment aligned with the vulnerability type and description.

Rule-based filtering. Subsequently, rule-based filtering is applied to assess the format and content of the responses. The main rules are as follows: (1) We perform a comparison between the responses and the ground truth to ensure answer correctness. (2) Samples lacking intermediate reasoning steps are filtered out by using regular expressions, as they do not reflect the reasoning process. (3) We enforce a strict response format, where tags such as "`<answer>...</answer>`" denote the final answer and "`<think>...</think>`" indicate the reasoning process; Samples not conforming to this format are discarded. The filtered dataset is denoted as $\mathcal{D}_{\text{vul}}$. The detailed statistical results are listed in Table s1 on GitHub.

c) Domain-aware SFT Training: Due to the limited vulnerability data availability, we leverage a dataset of challenging algorithmic optimization problems by CodeForce [41] (denoted as $\mathcal{D}_{\text{code}}$) to enhance the model's code reasoning capabilities. Unlike the single-task vulnerability repair setting, we construct a mixed dataset $\mathcal{D}_{\text{mixed}} = \mathcal{D}_{\text{vul}} \cup \mathcal{D}_{\text{code}}$. Each input includes a question x and additional contextual information i of vulnerability details for AVR, whereas algorithmic optimization problems include test cases. We utilize the CoT reasoning answers generated in the previous RAC step, which is denoted as the solution steps c. The answer serves as the ground truth y. The model is then trained to fit these reference responses, comprising both the intermediate steps c and the ground truth y by maximum likelihood estimation as below:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x,i,c,y)\sim\mathcal{D}_{\text{mixed}}} \log \pi_\theta \left(c, y \mid x, i\right). \quad (1)$$

C. Curriculum-based Verifiable Rewarded Training (CVRT)

This module is designed to train LLMs to acquire vulnerability-related reasoning capabilities and find the best generated answer by dynamically constructing a verifiable reward signal. It consists of three components: (a) reward design, (b) the RLVR paradigm, and (c) two-stage RLVR.

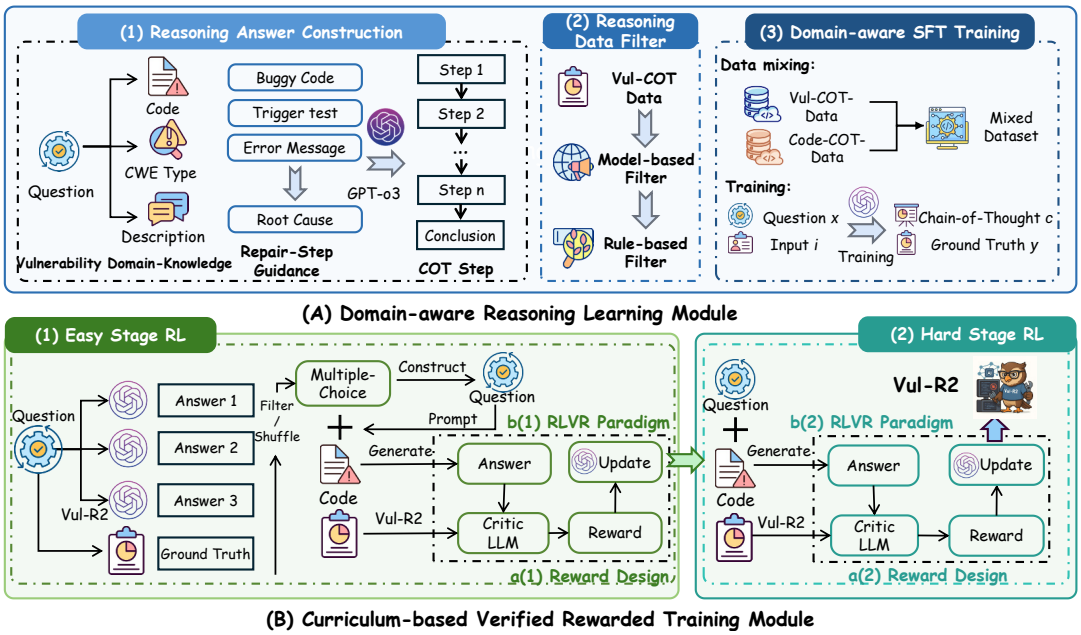


Fig. 2: The overview of Vul-R2.

(1) Instructions:
If you are a software engineer tasked with repairing vulnerabilities for {Improper Input Validation}, the description is {The product receives input or data, but it does not validate or incorrectly…}. You can delete, update, or insert code inside.\n

(2) Repair-step guidance:
You need to first analyse the buggy code, trigger test and error message. Then analyse the root cause of buggy code.
{Vulnerable Code}

(3) Reasoning specifications:
Think through the question step by step, enclose your reasoning process in <think>...</think> tags.

Fig. 3: The illustration of the prompt in the RAC. Contents in "{ }" will be substituted by the corresponding data.

Notably, the two stages are optimized through the designed reward and RLVR paradigm in the first two components. The complete process is presented in Algorithm 1.

a) Reward Design: Reward functions play a primary role in RLVR, shaping the landscape of policy optimization. Prior work [42] has demonstrated that appropriately adjusting task difficulty is crucial for promoting training. To avoid reward hacking [43], we design answer and format rewards.

To evaluate the correctness of the generated answer o, we employ an auxiliary LLM (i.e., Qwen2.5-14B-Instruct-1M [44]) as a critic to assess whether the proposed answer effectively fixes the vulnerability. It mitigates the influence of irrelevant factors such as whitespace, comments, or unrelated variable names for evaluation. Specifically, the answer reward

(1) LLM critic in RL stage:
Please determine if the following prediction answer correctly solves the question as ground truth does, and answer between "yes" (correct) or "no" (wrong):
Question: {Question + Code}
Ground Truth: {Ground Truth} \n Prediction Answer: {Prediction}
Please only answer "yes" or "no"

(2) Easy RL stage:
If you are a software engineer tasked with repairing vulnerabilities, generate fixed code to substitute each code segment enclosed by // bug_start and // bug_end.
Which the correct code can repair the vulnerability?
A) current = EMPTY;
B) if (tree[current].parent != EMPTY)
current = tree[current].parent;
C) build_cmap(ctx, cmap);
D) tree = cmap->tree;

Ground Truth: {tree = cmap->tree;}

(3) Hard RL stage:
Now generate fixed code to substitute each code segment enclosed by // bug_start and // bug_end."?

Ground Truth: {tree = cmap->tree;}

Fig. 4: The illustration of the prompt in the CVRT module.

is calculated as follows:

$$\mathcal{R}(o)_{\text{acc}} = \begin{cases} -2, & \text{if the "Critic" is 0,} \\ 1 + Sim(o, \text{GT}), & \text{otherwise.} \end{cases} \quad (2)$$

where Sim reflects the closeness of the generated repair to the human-written ground truth patch (i.e., GT) [45], which ranges from 0 to 1. The critic score (i.e., "Critic") of 1 is assigned if the critic LLM confirms that the answer correctly repairs the vulnerability. Formally, the critic LLM's prompt is structured as in Fig. 4 (1).

Algorithm 1 The curriculum-based verifiable rewarded training process.

Input : The original question X, the vulnerable code needs to be repaired, and the corresponding contextual information I, Answer Y, Model θ, maximum generation number J.

1 **Function** *Curriculum-based Verifiable Rewarded Training*:
2 // Easy Stage RLVR
3 **for** *all* $x_n \in X$ **do**
4 Obtain question x_n, context input i_n, and ground truth y_n
5 j = 0
6 **while** $x < J$ **do**
7 Generate answer x_j via θ
8 **if** *Format correct* **then**
9 j += 1
10 **end**
11 **end**
12 Select answers samples x_j and construct prompt x_{easy} in Fig. 3 (5)
13 Optimization Process by Eq. 3, 5, 6, and 7 with x_{easy}, i_n, y_n
14 Update Model θ
15 **end**
16 // Hard Stage RLVR
17 **for** *all* $x_n \in Q$ **do**
18 Obtain question x_n, context input i_n, and ground truth y_n
19 Construct prompt x_{hard} in Fig. 3 (6)
20 Optimization Process by Eq. 3, 5, 6, and 7 with x_{hard}, i_n, y_n
21 Update Model θ
22 **end**
23 **return** *Model* θ

The format reward enforces output consistency and enhances the interpretability and post-processing of the model's responses. It takes the value of 1 if the model's output strictly follows this predefined format. If the required tags are missing or incorrectly used, the format reward is set to -1.

b) RLVR Paradigm: We use the same paradigm as inspired by REINFORCE++ [46] for two training phases. This can be viewed as a curriculum approach between easy and hard questions. To enhance training stability and efficiency between the two stages, we employ the KL divergence in the training objectives. Specifically, the objective function for sequence generation is formulated as follows:

$$\begin{aligned}\mathcal{J}_{\text{Re++}}(\theta) = \mathbb{E}_{[q\sim P(Q),\{o_i\}_{i=1}^{N}\sim\pi_{\theta_{\text{old}}}(O|q)]} \\ \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left[\frac{\pi_\theta^{i,t}}{\pi_{\theta_{\text{old}}}^{i,t}}\hat{A}_{i,t},\right.\right. \\ \left.\text{clip}\left(\frac{\pi_\theta^{i,t}}{\pi_{\theta_{\text{old}}}^{i,t}}, 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}\right] \\ \left.-\beta\mathbb{D}_{\text{KL}}[\pi_\theta\|\pi_{\text{ref}}]\right\}\end{aligned} \tag{3}$$

where $P(Q)$ represents the distribution of vulnerability repair questions used during training, with q denoting a sample drawn in the current training iteration. The old policy and the current policy of base model are represented by $\pi_{\theta_{\text{old}}}$ and $\pi_{\theta_{\text{new}}}$, respectively, where o corresponds to a complete response sampled from the respective policy. The reference policy, corresponding to the frozen base model parameters, is denoted by $\pi_{\theta_{\text{ref}}}$. Additionally, N indicates the number of responses sampled per question in each iteration and ϵ is the clipping threshold for policy updates. $\hat{A}_{i,t}$ denotes the advantage function, which is defined as:

$$\hat{A}_{i,t} = \frac{r_i - \text{mean}(\{r_1, r_2, \cdots, r_N\})}{\text{std}(\{r_1, r_2, \cdots, r_N\})} \tag{4}$$

$$r_i = r(x, y) - \beta \cdot \sum_{i=t}^{T}\mathbb{D}_{\text{KL}}\left[\pi_\theta\|\pi_{\text{ref}}\right] \tag{5}$$

where $\{r_1, r_2, \cdots, r_N\}$ denotes a group of rewards corresponding to the outputs within each group. $\mathbb{D}_{\text{KL}}\left[\pi_\theta\|\pi_{\text{ref}}\right]$ denotes the KL divergence [47], which employs an unbiased estimator and is formulated as follows:

$$\mathbb{D}_{\text{KL}}\left[\pi_\theta\|\pi_{\text{ref}}\right] = \frac{\pi_{\text{ref}}\left(o_{i,t}|q, o_{i,<t}\right)}{\pi_\theta\left(o_{i,t}|q, o_{i,<t}\right)} - \log\frac{\pi_{\text{ref}}\left(o_{i,t}|q, o_{i,<t}\right)}{\pi_\theta\left(o_{i,t}|q, o_{i,<t}\right)} - 1. \tag{6}$$

where $\frac{\pi_\theta(o_{i,t}|q)}{\pi_{\theta_{\text{ref}}}(o_{i,t}|q)}$ is the policy ratio. It regularizes the policy update, ensuring that π_θ does not deviate excessively from the reference model $\pi_{\theta_{\text{ref}}}$.

c) Two Stage RLVR: Directly applying the existing RLVR paradigm to software vulnerability repair is generally ineffective. This is primarily due to the the vulnerability repair process does not produce verifiable intermediate feedback, which hinders its ability to generate accurate vulnerability fixes. Specifically, we propose a two-stage RLVR process comprising an easy stage and a hard stage, as detailed in Algorithm 1. **(1) Easy Stage:** We reformulate the generation task as a multiple-choice problem. Specifically, to initialize the easy stage RL (Lines 3-15 in Algorithm 1). Each prompt samples up to J from the current dataset $\mathcal{D}_{\text{vul}}$ as candidate vulnerability repair answers. These approximate incorrect answers x_j, together with the ground truth, constitute options A, B, C, and D to construct multiple-choice prompt x_{easy} in Fig. 4 (2). The model θ is then instructed to output the single-letter choice, along with the corresponding ground truth y enclosed within "`<answer>...</answer>`" tags. This approach encourages the model to explore solution paths that are similar to the correct answer, thereby reducing the likelihood of generating entirely invalid responses. Importantly, the model is not limited to generating only the single-letter choice; it also produces the complete repair code, which is valuable for the subsequent training stage. **(2) Hard Stage:** In this stage, we continue training the model that was previously trained in the easy stage, without providing multiple-choice prompts, allowing the model to perform the vulnerability repair task in a more open-ended manner. During this phase, we encourage the model to engage in unrestricted exploration, which facilitates the acquisition of broader knowledge. The detailed prompt is shown in Fig. 4 (3). The prompts used in this stage are consistent with those employed during the inference phase, comprising the question x_{hard}, input i, and ground truth y components. The whole process is elaborated in our Algorithm 1. The detailed data statistics are provided in Table s2 on the GitHub repository.

III. Experimental setup

A. Research Questions

In this section, we evaluate the effectiveness of Vul-R2 by comparing it with the state-of-the-art baselines and focus on the following six research questions (RQs):

- **RQ1:** How effective is Vul-R2 compared with existing vulnerability repair approaches?
- **RQ2:** How effective is Vul-R2 in repairing vulnerabilities across different vulnerability types?
- **RQ3:** How do RLVR techniques contribute to the performance of Vul-R2?
- **RQ4:** What is the impact of incorporating different reasoning data during the SFT phase?
- **RQ5:** What is the influence of different components of Vul-R2 on the performance for repairing vulnerabilities?
- **RQ6:** What is the influence of hyperparameters on the performance of Vul-R2?

B. Datasets

In this paper, we focus on C/C++ programs due to their widespread adoption in real-world software development and the prevalence of well-known vulnerabilities that have been accurately labeled by security researchers. Specifically, we utilize two widely-used and high-quality datasets: PrimeVul [35] and SVEN [36].

a) PrimeVul: PrimeVul is a large-scale real-world C/C++ dataset. It aggregates vulnerability-related commits from four established datasets, BigVul [48], CVEFixes [49], CrossVul [50], and DiverseVul [51], covering more than 140 CWE categories. It applies a stringent deduplication process and adopts a temporally-aware data splitting strategy, resulting in training, validation, and test sets containing 3,789, 480, and 435 samples, respectively.

b) SVEN: SVEN dataset manually vets vulnerabilities from multiple repositories of C/C++ code. It consists of 384 C/C++ vulnerabilities, covering 9 CWEs. By manual inspection, the accuracy of SVEN reaches 94.0%. We use SVEN as the test set without further training.

C. Baselines

To provide a comprehensive evaluation, we experiment on two types of methods, with details as below. *(a) CodePTM-based methods:* We choose two recent CodePTM-based works, VulRepair [25] and VulMaster [26], as baselines. VulRepair fine-tunes CodeT5 with BPE tokenization [52] to improve automated patch generation accuracy. VulMaster incorporates CWE knowledge and code structure into CodeT5 for AVR. *(b) LLM-based methods:* We select three large open-source LLMs: Llama3-70B-Instruct [53], Qwen2.5-14B-Instruct-1M [44], and Qwen2.5-Coder-32B-Instruct [54], due to their strong performance in code generation tasks. In addition, we incorporate the closed-source LLM OpenAI-o3 [38] for vulnerability repair, given its demonstrated effectiveness in handling general tasks.

D. Metrics

Following the previous methods [25], [26], we employ the EM, Success, and CodeBLEU metrics as evaluation metrics:

a) Exact Match and Success: Exact match (EM) is the percentage of generated fixes that match the token sequence of the ground truth, which means this part of the fixes is correct. It is formulated as follows: $EM = \frac{\text{Correct Fixes}}{\text{Vulnerabilities}}$. Success metric equals the number of vulnerabilities that are successfully fixed.

b) CodeBLEU: CodeBLEU [55] is used to evaluate the similarity between the generated code and the ground truth. It's a variant of BLEU [56] specialized for code tasks, which also considers the structural similarity of source code.

E. Implementation Details

During the inference phase, we set the maximum beam size to 10 for most baseline implementations, except for OpenAI-o3 [38], where the beam size is set to 5. This choice is reasonable, as our beam size is comparable to or smaller than those used in prior work [57], [58].

In the DARL module, we employ OpenAI-o3 [38] as the data generation model for the RAC phase. For the SFT phase, we train Qwen-14B-Instruct-1M [44] with a batch size of 16, using a constant learning rate of $1e^{-4}$ and the AdamW optimizer. The maximum prompt and response lengths are both set to 4,096 tokens. Additional training settings include gradient accumulation steps of 16, a cosine learning rate scheduler, three training epochs, and a warmup ratio of 0.1.

For the CVRT phase, the training batch size is set to 16, with a maximum prompt length of 1,024 and a maximum response length of 4,096. The actor's learning rate is set to $2e^{-6}$. The rollout temperature is set to 1.0. We generate 8 and 16 rollouts per question for RL reward computation in the easy and hard stages, respectively.

During the DARL stage, we utilized 8 NVIDIA H20 GPUs. For the CVRT stage, we employed 16 H20 GPUs for training. The detailed training set is available at GitHub.

IV. Experimental Results

A. RQ1: Comparison with SOTA

To assess the effectiveness of Vul-R2 in improving the performance of vulnerability repair, we compare it with seven baselines. Table I presents the performance of Vul-R2 with baselines on PrimeVul and SVEN datasets.

1) Vul-R2 vs CodePTMs: The results summarized in Table I demonstrate that Vul-R2 consistently outperforms all SFT-based CodePTMs methods across both evaluation datasets. Specifically, Vul-R2 achieves an average improvement of 28.00% in EM and 15.33% in CodeBLEU, respectively. Furthermore, on the SVEN dataset, where no additional training is conducted, Vul-R2 is able to correctly repair 144 vulnerabilities and achieves an EM of 39.13%. In comparison, Vulmaster and VulRepair are only able to repair 27 and 8 vulnerabilities, respectively. These results indicate that the reasoning-based method adopted by Vul-R2 demonstrates stronger generalization capability on previously unseen datasets, which contributes to its superior performance.

TABLE I: Experimental results of Vul-R2 and the vulnerability repair baselines on the PrimeVul and SVEN datasets. Texts in bold represent the best performance of the best methods in each metric.

Type	Method	Dataset		PrimeVul [35]			SVEN [36]		
		Paradigm	Sample Size	Success↑	Exact Match ↑	CodeBLEU↑	Success↑	Exact Match↑	CodeBLEU↑
CodePTM	VulMaster	SFT	-	20	4.59	36.74	27	7.34	47.45
	VulRepair	SFT	10	8	1.84	32.86	8	2.17	36.77
LLM	Qwen2.5-14B-Instruct	COT	10	43	9.89	35.89	42	11.41	41.73
	Qwen2.5-32B-Instruct	COT	10	51	11.72	34.79	45	12.23	41.45
	LLama3-70B-Instruct	COT	10	24	5.52	34.47	7	3.95	40.33
	OpenAI-o3	COT	5	32	7.36	27.32	30	8.17	38.78
	Qwen2.5-14B-Instruct	SFT	10	59	13.56	39.20	112	30.35	56.68
Ours	**Vul-R2***	RLVR	10	**103**↑44	**23.67**↑10.11%	**42.95**↑3.75%	**142**↑30	**38.59**↑8.24%	**60.07**↑3.39%
	Vul-R2	SFT & RLVR	10	**108**↑49	**24.83**↑11.27%	**46.17**↑6.97%	**144**↑32	**39.13**↑8.78%	**61.40**↑4.72%

TABLE II: Number of exact matches for the fine-grained CWE vulnerability types.

CWE-Type Category	VulMaster	OpenAI-o3	Vul-R2	EM ↑
Resource Management Error	9/229	15/229	59/229	25.76+19.21%
Improper Check or Handling of Exceptional Conditions	0/52	1/52	9/52	17.31+15.38%
NULL Pointer Dereference	2/39	6/39	11/39	28.21+12.82%
Numeric Error	4/35	4/35	10/35	28.57+17.14%
Insufficient Control Flow Management	3/26	0/26	8/26	30.77+19.23%
Improper Input Validation	0/24	1/24	4/24	16.67+12.50%
Improper Access Control	1/19	4/19	6/19	31.58+10.53%
Injection	0/4	0/4	0/4	0.00+0%
Others	1/11	1/11	1/11	9.09+0%

2) Vul-R2 vs LLMs: As shown in Table I, Vul-R2 consistently outperforms all LLM-based methods across all datasets and evaluation metrics, even when compared to generally robust models such as OpenAI-o3. Specifically, Vul-R2 achieves an EM score of up to 24.83% and a CodeBLEU score of 46.17% on the PrimeVul dataset. Moreover, on the SVEN dataset, Vul-R2 exhibits a higher EM of 39.13% and CodeBLEU of 61.40%, further demonstrating its superior performance. There are two primary factors that may contribute to this performance gap. First, the reasoning-based paradigm adopted by Vul-R2 generates a multi-step, verifiable, and high-quality repair process, which is ignored by previous paradigms such as CoT prompting or SFT. Second, the LLM-based approaches may lack sufficient domain-specific knowledge required for effective vulnerability repair, making it particularly challenging to detect and address complex vulnerability patterns, even when advanced reasoning capabilities are present in models such as OpenAI-o3.

Answer to RQ1: Vul-R2 achieves the best overall performance across all evaluated metrics, with improvements of 8.78∼36.96% in EM and the identification of 32∼137 additional vulnerabilities on the PrimeVul and SVEN datasets.

B. RQ2: Comparison with Vulnerability Types

To evaluate the effectiveness of Vul-R2 in repairing different types of vulnerabilities, we select nine categories and 62 types of CWE across 143 projects. The proportion and types of vulnerabilities are presented in Table II, and the selection criteria are based on CWE [39]. It is important to note that each category may contain multiple specific CWE types.

Overall, we observe that Vul-R2 is effective across all the categories of vulnerabilities analyzed, achieving an average performance improvement of 18.04% in EM. Specifically, we can observe the following findings: (1) Vul-R2 demonstrates excellent performance in repairing vulnerabilities associated with Numeric Errors, correctly repairing 10 out of 35 vulnerabilities, resulting in an accuracy of 28.57%. This could be attributed to the RLVR paradigm, which excels at diverse reasoning skills under verifiable rewards, including arithmetic and logic [59]. (2) In addition, for the majority of vulnerability categories, Vul-R2 repairs more vulnerabilities than baselines. For example, in categories such as NULL Pointer Dereference, Improper Access Control, and Resource Management Error, Vul-R2 achieves accuracy rates of 28.21%, 31.58%, and 25.76%, respectively. (3) For vulnerability categories with limited training data, such as Injection vulnerabilities, Vul-R2 finds it challenging to achieve performance improvements. This is primarily because the same type of vulnerability can be triggered through various mechanisms, making it difficult to enhance performance when data is scarce.

TABLE III: Performance of Vul-R2 with respect to different RLVR-based methods.

Method	PrimeVul [35]			SVEN [36]		
	Success ↑	EM↑	CodeBLEU ↑	Success ↑	EM↑	CodeBLEU ↑
Vul-R2	**108**	**24.83**	**46.17**	**144**	**39.13**	**61.40**
- *w/ GRPO*	98↓-10	22.53↓-2.30%	43.79↓-2.38%	135↓-9	36.68↓-2.45%	60.57↓-0.83%
- *w/o Critic LLM*	101↓-7	23.22↓-1.61%	45.00↓-1.17%	143↓-1	38.86↓-0.27%	61.17↓-0.23%
- *w/ zero-RLVR*	103↓-5	23.67↓-1.16%	42.95↓-3.22%	142↓-2	38.59↓-0.54%	60.07↓-1.33%

Answer to RQ2: Across most vulnerability categories, Vul-R2 successfully repairs more cases than other baseline methods. Vul-R2 particularly demonstrates strong performance in repairing vulnerabilities such as Numeric Errors.

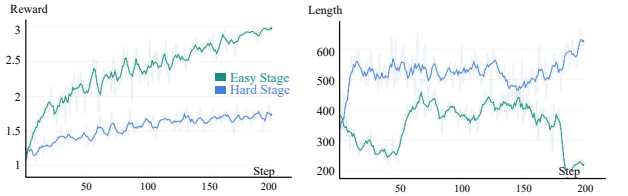
C. RQ3: Analysis of RLVR

To answer this RQ, we investigate the impact of the RLVR in Vul-R2 on AVR performance, and explore whether RLVR can trigger "aha moment" [60] similar to those observed in DeepSeek-R1. Due to space limitations, we present only a subset of the experimental results and training curves.

1) Effectiveness of RLVR: We design three variants: (1) a variant that replaces the CVRT component with GRPO, which is the RLVR method employed by DeepSeek-R1 [60]) (i.e., *- w/ GRPO*). (2) a variant of Vul-R2 in which the CVRT operates without a critic LLM. Instead, the reward signal is computed solely based on rule-based correctness criteria (i.e., -

TABLE IV: Performance of Vul-R2 with respect to different reasoning data domains.

Data Type	PrimeVul [35]			SVEN [36]		
	Success ↑	EM↑	CodeBLEU ↑	Success ↑	EM↑	CodeBLEU ↑
- *w/o Reason*	59	13.56	39.20	112	30.35	56.68
- *w/ Reason*	75$^{\uparrow 16}$	17.24$^{\uparrow 3.68\%}$	39.20$^{0.00\%}$	106$^{\downarrow 6}$	28.80$^{\downarrow 1.55\%}$	54.41$^{\downarrow 2.27\%}$
- *w/ Reason & Math*	79$^{\uparrow 20}$	18.16$^{\uparrow 4.60\%}$	36.49$^{\downarrow 2.71\%}$	108$^{\downarrow 4}$	29.35$^{\downarrow 1.00\%}$	47.55$^{\downarrow 9.13\%}$
- *w/ Reason & Code*	**97**$^{\uparrow 38}$	**22.30**$^{\uparrow 8.74\%}$	**43.87**$^{\uparrow 4.67\%}$	**137**$^{\uparrow 25}$	**37.23**$^{\uparrow 6.88\%}$	**61.12**$^{\uparrow 4.44\%}$



(a) Reward (b) Response Length

Fig. 5: Reward and mean response length during RLVR training (Vul-R2*), illustrating how the model autonomously learns to allocate more thinking compute.

w/o Critic LLM). (3) a variant that applies the RLVR paradigm directly, by passing the initial SFT phase (i.e., - *w/ zero-RLVR*).

As shown in Table III, the CVRT component consistently outperforms GRPO across 2.3% of EM and 2.38% of CodeBLEU in PrimeVul. This observation is consistent with prior findings reported by Xie et al. [42]. Additionally, we find that the inclusion of a critic LLM has a substantial impact on performance. This may be attributed to the inherently diverse and non-deterministic nature of vulnerability repair, where multiple correct solutions often exist. A critic LLM is capable of assessing finer-grained aspects of generated patches, that rule-based metrics tend to overlook, thereby providing more discriminative reward signals during training. Finally, we observe that even when used RLVR paradigm independently, the RLVR paradigm yields competitive results, achieving an EM of 31.13% and a CodeBLEU score of 51.51%. These results collectively highlight the effectiveness of the RLVR paradigm in enhancing the reasoning capabilities of LLMs for AVR.

2) Aha Moment in AVR: We observe clear signs of "aha moment", as previously described by DeepSeekAI [60], wherein Vul-R2 demonstrates emergent reasoning abilities during the vulnerability repair process. To the best of our knowledge, this is the first empirical study to reveal such phenomena within the context of real-world, vulnerability-related tasks, extending the findings of DeepSeek-R1 [60].

As illustrated in Fig. 5 (a), the easy stage learn the reward signal more effectively. Performance in both stages improves progressively during model training. In Fig. 5 (b), the Vul-R2* leads to increased average response lengths in the hard stage (i.e., blue lines), with the response length rising from 300 to 500. It suggests that the model allocates more "thinking time" to reflect upon its initial assumptions before arriving at a final repair. This reflective behavior emerges through interaction with the RLVR process and is not prompted by explicit instructions. We also observe a noticeable rise in the usage of introspective linguistic markers, such as "verify" and "ensure", which often indicate nuanced reasoning. Such patterns suggest the development of a more deliberate reasoning process.

TABLE V: Impact of model selection for reasoning answer construction on Vul-R2's performance.

Variant	PrimeVul [35]			SVEN [36]		
	Success ↑	EM↑	CodeBLEU ↑	Success ↑	EM↑	CodeBLEU ↑
- *w/ Qwen2.5-14B*	75	17.24	39.20	106	28.80	54.41
- *w/ Qwen2.5-32B*	79	18.16	36.49	108	29.35	47.55
- ***w/ OpenAI-o3***	**97**$^{\uparrow 18}$	**22.30**$^{\uparrow 4}$	**43.87**$^{\uparrow 7.38\%}$	**137**$^{\uparrow 29}$	**37.23**$^{\uparrow 7}$	**61.12**$^{\uparrow 13.57\%}$

Answer to RQ3: Our results provide the first empirical evidence that the RLVR technique improves the performance of reasoning capabilities in vulnerability-related tasks, as indicated by the emergence of an observable "aha moment" within the vulnerability domain.

D. RQ4: Analysis of Reasoning Data

To systematically explore the contribution of the reasoning data within Vul-R2, we analyze the impact of two key factors on its performance: (1) the domain characteristics of the reasoning data, and (2) the selection of the model employed to generate the reasoning answer.

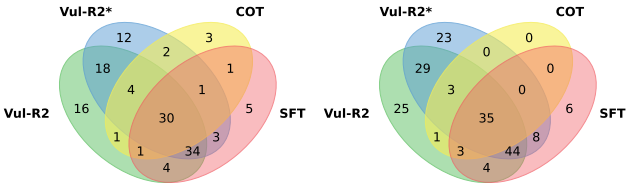
1) Domain Characteristics: Table IV investigates the impact of reasoning data domain characteristics on the effectiveness of vulnerability repair. We conduct the following variants: (1) using only the original dataset as a baseline (i.e., *w/o Reason*), (2) augmenting the dataset with reasoning answers generated by RAC (i.e., *w/ Reason*), (3) further enriching the data in (2) with mathematical domain samples from AMC [61] and AIME [62] (i.e., *w/ Reason & Math*), and (4) further enriching the data in (2) with code domain samples from Codeforces [41] (i.e., *w/ Reason & Code*).

As shown in Table IV, for both PrimeVul and SVEN, Vul-R2 achieves optimal performance when the reasoning data is constructed using RAC in combination with code domain data. This configuration yields improvements of 8.74% and 6.88%, respectively, in the EM metric. In contrast, incorporating the training data with the mathematical domain does not yield additional performance gains. It leads to a reduction in CodeBLEU scores. We hypothesize that this is because code and vulnerabilities share greater formal and semantic similarity, whereas the addition of mathematical domain data may introduce redundancy, potentially hindering the model's ability to effectively repair vulnerabilities.

2) Model Selection: We evaluate the performance of Vul-R2 using reasoning answers generated by different models, including Qwen-14B-Instruct-1M [44] (i.e., *w/ Qwen2.5-14B*), Qwen-Coder-32B-Instruct [54] (i.e., *w/ Qwen2.5-32B*), and OpenAI-o3 [38]. The corresponding results are presented in Table V. From these results, we observe that Vul-R2 achieves progressively better performance when utilizing reasoning data generated by more advanced models, with the best results obtained using OpenAI-o3. Furthermore, the quality of the reasoning data significantly impacts overall performance, which we attribute to the varying degrees of hallucination exhibited

TABLE VI: Ablation Study.

Variant	PrimeVul [35]			SVEN [36]		
	Success ↑	EM↑	CodeBLEU ↑	Success ↑	EM↑	CodeBLEU ↑
- *w/o DAC*	59$^{\downarrow -49}$	13.56$^{\downarrow -11.27\%}$	39.20$^{\downarrow -6.97\%}$	112$^{\downarrow -32}$	30.35$^{\downarrow -8.78\%}$	56.68$^{\downarrow -4.72\%}$
- *w/o CVRT*	97$^{\downarrow -11}$	22.30$^{\downarrow -2.53\%}$	43.87$^{\downarrow -2.30\%}$	137$^{\downarrow -7}$	37.23$^{\downarrow -1.90\%}$	61.12$^{\downarrow -0.28\%}$
Vul-R2	**108**	**24.83**	**46.17**	**144**	**39.13**	**61.40**



(a) PrimeVul. (b) SVEN.

Fig. 6: Venn diagram of the number of successfully fixed vulnerabilities for *COT*, *SFT*, Vul-R2* (i.e., *RLVR*), and Vul-R2 (i.e., *SFT & RLVR*) four paradigms.

by the underlying models. Therefore, we recommend employing more powerful foundation models to generate reasoning answers for optimal results.

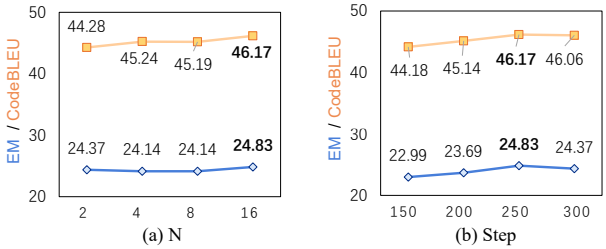
Answer to RQ4: Optimal AVR performance is achieved when reasoning data is incorporated with the code domain, while incorporating mathematical domain data does not yield further improvements. The effectiveness of reasoning data is highly dependent on the data quality, with stronger base models such as OpenAI-o3 producing superior results.

E. RQ5: Ablation Study

To assess the contributions of key components within Vul-R2, we conduct a comprehensive ablation study, with the results summarized in Table VI. Specifically, we implement two variants of Vul-R2: (1) one without the reasoning data generated by reasoning answer construction (i.e., *w/o RAC*), and (2) another without the curriculum-based verifiable rewarded training module (i.e., *w/o CVRT*), wherein only the SFT stage is retained.

As shown in Table VI, the removal of either component leads to a noticeable degradation in performance. In particular, excluding the reasoning data results in an average performance drop of 10.03% in EM and 5.85% in CodeBLEU across both evaluated datasets. Similarly, omitting the CVRT module yields a decline of 2.22% in EM , highlighting the importance of curriculum-based reinforcement learning in enabling LLMs to learn from previously unexplored generations and enhance their generalization capability.

In addition, we conduct comparative experiments on four paradigms: COT, SFT, Vul-R2* (RLVR), and Vul-R2 (*SFT & RLVR*). As shown in Fig. 6, Vul-R2 and Vul-R2* outperform the previous methods, successfully repairing 16 and 12 unique vulnerabilities on PrimeVul, respectively. This finding suggests that approaching the vulnerability repair task from a reasoning perspective through RLVR training can yield better results.



(a) N (b) Step

Fig. 7: Parameter analysis.

Answer to RQ5: Each component of Vul-R2 plays a critical role in achieving optimal performance. Furthermore, we validate the effectiveness of the RLVR paradigm for AVR.

F. RQ6: Parameter Analysis

In this section, we investigate the influence of two parameters: the number of generated examples (N) in RLVR and the training step, on the performance of Vul-R2 in the PrimeVul.

a) Number of Generated Examples by RLVR: We study the effect of N by varying it from 2 to 16. As depicted in Fig. 7 (a), Vul-R2 exhibits optimal performance when N is set to 16, with larger values yielding improved results. We suggest that this is because selecting more samples increases the exploration space, making it easier to obtain correct rewards in RLVR. However, increased exploration also leads to higher computational overhead.

b) Number of Training Steps: We evaluate the performance of Vul-R2 across different training steps in RLVR. The results are presented in Fig. 7 (b). We find that Vul-R2 achieves the best performance at 250 steps, as further increasing the number of training steps may introduce the risk of overfitting problems due to the limited data. Consequently, we set the maximum number of iterations to 250 steps.

Answer to RQ6: The performance of Vul-R2 improves as more samples are generated; however, large numbers lead to increased computational resource consumption. Vul-R2 achieves its best performance with 250 training steps.

V. Discussion

A. What makes Vul-R2 work?

a) Empowering LLMs with vulnerability-specific knowledge for AVR: Vul-R2 can effectively equip LLMs with explicit knowledge of vulnerability patterns. As illustrated in Fig. 8, this example demonstrates a case of "NULL Pointer Dereference" where all baseline methods failed to correctly fix the vulnerability, whereas Vul-R2 successfully repairs it. LLMs primarily rely on general knowledge acquired during pre-training. Consequently, without explicit training on vulnerabilities, LLMs often struggle to repair vulnerabilities in real-world scenarios. In contrast, Vul-R2 accurately captures the vulnerability pattern in this case, the code fails to check whether the number of parameters is sufficient, which may lead to out-of-bounds access or dereferencing a NULL pointer.

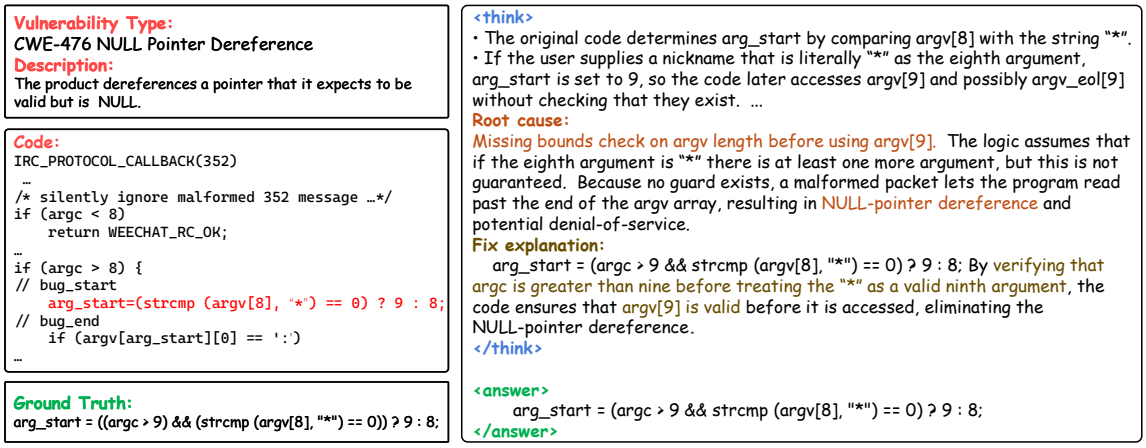


Fig. 8: **Case Study**. A case of CVE-2020-9759 [63] showing that Vul-R2 can accurately repair the vulnerability with detailed, interpretable reasoning answer, as illustrated in the right-side figure.

By explicitly modeling and leveraging vulnerability patterns, rather than focusing solely on code semantics, Vul-R2 demonstrates superior effectiveness.

b) More detailed reasoning process by CVRT empowers LLMs for more effective AVR: The second advantage of Vul-R2 lies in its RLVR-based CVRT module, which enhances the reasoning capabilities of LLMs by dynamically constructing verifiable reward signals into the training process. In contrast, prior works [25], [26] typically generate vulnerability patches directly, without a reasoning process. As illustrated in Fig. 8, Vul-R2 first analyzes the original code to identify existing issues. Then, it pinpoints the root cause of a vulnerability, such as the "missing bounds check on argv length before using argv[9]" (mentioned in "Root cause"). Subsequently, Vul-R2 generates a detailed fix explanation, which includes "verifying" the proposed solution and the steps taken to "ensure" its correctness (mentioned in "Fix explanation"). Vul-R2 systematically trains LLMs through each stage of reasoning, with the process being guided by verifiable feedback. Our results demonstrate that the strong reasoning capabilities enabled by CVRT improve vulnerability repair.

B. Threats to Validity

We have identified the following threats and limitations:

a) Generalizability to Other Programming Languages: In this work, we conduct experiments on the PrimeVul and SVEN datasets, which cover two widely-used programming languages: C and C++. Although additional datasets exist for other languages, such as the Java dataset in the Vul4J [64] Benchmark. We do not include them in our evaluation due to an insufficient number of samples to support the training objectives of RLVR. In future work, we plan to continue collecting data and to extend our experiments to encompass a broader range of programming languages.

b) Constraints on Reasoning Context Length: All reasoning samples in our experiments are less than 4,096 tokens in length. Consequently, Vul-R2 may encounter difficulties in repairing vulnerabilities within code snippets that exceed this length, potentially resulting in the loss of relevant contextual information due to truncation. This limitation may affect the evaluation of longer code segments in the context of software vulnerability repair. We plan to address this issue in future work by conducting experiments with increased computational resources to accommodate longer input sequences.

c) Limitations in LLM Selection: Another potential threat to validity stems from the selection of foundation models used in Vul-R2. Following prior work [42], we evaluate Vul-R2 using the Qwen-2.5 series [54]. In future work, we intend to further assess the effectiveness of Vul-R2 across a wider variety of LLMs.

VI. Related Work

A. Automated Vulnerability Repair

Previous research has proposed various methods for AVR, which can be broadly categorized into supervised learning-, CodePTM-, and LLM-based methods. Supervised learning-based approaches [25], [26], [65]–[67] learn to generate patches by observing patterns in labeled vulnerability-fix pairs. For example, Vurle [66] represents one of the earliest learning-based frameworks, which learns contextual code transformations directly from examples of vulnerable code and their corresponding fixes. CodePTMs are neural models pretrained on large-scale corpora of code and are often fine-tuned on curated datasets of vulnerability fixes. For example, VulRepair [25] fine-tunes CodeT5 using a byte pair encoding tokenizer [52] and the CVEFixes dataset [49]. Similarly, VulMaster [26] also builds upon CodeT5, augmenting the model with abstract syntax trees and CWE examples generated by ChatGPT [24]. LLM-based approaches [53], [54], [68] use zero-shot or few-shot prompting techniques to AVR. For example, Wu et al. [69] conducted evaluations of LLMs on the Vul4J dataset [64].

B. RLVR and Software Engineering

RLVR has recently emerged as a promising method for enhancing the reasoning capabilities of LLMs [42], [45], [60], [70], [71]. For instance, DeepSeek-AI introduced GRPO and DeepSeek-R1 [60], showcasing the effectiveness of RLVR in improving LLM reasoning performance. SWE-RL [45] pioneered the application of reinforcement learning for LLM-based reasoning in real-world software development tasks. Furthermore, SRPO [71] proposed a cross-domain training framework that jointly optimizes mathematical reasoning and programming proficiency, demonstrating improved generalization across diverse task categories. Despite these advancements, our approach is the first to train LLMs by RLVR in the vulnerability domain, aiming to enhance the reasoning capabilities of LLMs without relying on intermediate feedback from the environment.

VII. Conclusion and Future Work

In this paper, we propose *Vulnerability Reasoner and Repair (Vul-R2)*, a reasoning LLM for vulnerability repair. It models the vulnerability repair task from a reasoning perspective, which comprises two key components: a domain-aware reasoning learning module and a curriculum-based verifiable rewarded training module. Vul-R2 enables the model to reason about complex vulnerability patterns and generate effective patch candidates for AVR. We conduct comprehensive experiments on the PrimeVul and SVEN datasets to evaluate the effectiveness of Vul-R2 in enhancing vulnerability repair capabilities. In the future, we plan to apply our reasoning LLM to other tasks in the vulnerability domain, such as vulnerability detection. Our source code is available at https://github.com/Xin-Cheng-Wen/Vul-R2.

Acknowledgment

This research is supported by the National Natural Science Foundation of China under project (No. 62472126, 62276075), Natural Science Foundation of Guangdong Province (Project No. 2023A1515011959), and Shenzhen-Hong Kong Jointly Funded Project (Category A, No. SGDX20230116 091246007).

References

[1] X. Wen, X. Wang, C. Gao, S. Wang, Y. Liu, and Z. Gu, "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, 2023, pp. 345–357.

[2] X. Wen, C. Gao, F. Luo, H. Wang, G. Li, and Q. Liao, "LIVABLE: exploring long-tailed classification of software vulnerability types," *IEEE Trans. Software Eng.*, vol. 50, no. 6, pp. 1325–1339, 2024.

[3] Statista, "Number of common it security vulnerabilities and exposures (cves) worldwide from 2009 to 2024 ytd," 2024. [Online]. Available: https://www.statista.com/statistics/500755/worldwide-common-vulnerabilities-and-exposures/

[4] S. Cao, X. Sun, L. Bo, R. Wu, B. Li, and C. Tao, "MVD: memory-related vulnerability detection based on flow-sensitive graph neural networks," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 1456–1468.

[5] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet?" *CoRR*, vol. abs/2009.07235, 2020.

[6] 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*, 2019, pp. 10 197–10 207.

[7] C. Wang, Z. Li, Y. Peng, S. Gao, S. Chen, S. Wang, C. Gao, and M. R. Lyu, "REEF: A framework for collecting real-world vulnerabilities and fixes," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023, pp. 1952–1962.

[8] "171 cyber security statistics: 2025's updated trends and data," 2025. [Online]. Available: https://keepnetlabs.com/blog/171-cyber-security-statistics-2024-s-updated-trends-and-data

[9] R. Croft, M. A. Babar, and M. M. Kholoosi, "Data quality for software vulnerability datasets," *CoRR*, vol. abs/2301.05456, 2023.

[10] Q. Gao, Y. Xiong, Y. Mi, L. Zhang, W. Yang, Z. Zhou, B. Xie, and H. Mei, "Safe memory-leak fixing for C programs," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, A. Bertolino, G. Canfora, and S. G. Elbaum, Eds. IEEE Computer Society, 2015, pp. 459–470.

[11] S. Hong, J. Lee, J. Lee, and H. Oh, "SAVER: scalable, precise, and safe memory-error repair," in *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 271–283.

[12] J. Lee, S. Hong, and H. Oh, "Memfix: static analysis-based repair of memory deallocation errors for C," 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*, G. T. Leavens, A. Garcia, and C. S. Pasareanu, Eds. ACM, 2018, pp. 95–106.

[13] Z. Li, Z. Liu, W. K. Wong, P. Ma, and S. Wang, "Evaluating C/C++ vulnerability detectability of query-based static application security testing tools," *IEEE Trans. Dependable Secur. Comput.*, vol. 21, no. 5, pp. 4600–4618, 2024.

[14] A. Inc, "Clang static analyzer," [n.d.]. [Online]. Available: https://clang-analyzer.llvm.org/scan-build.html

[15] Facebook, "Infer," [n.d.]. [Online]. Available: https://fbinfer.com/.

[16] J. Pewny, F. Schuster, L. Bernhard, T. Holz, and C. Rossow, "Leveraging semantic signatures for bug search in binary programs," in *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014*. ACM, 2014, pp. 406–415.

[17] Y. Yang, T. Zhou, K. Li, D. Tao, L. Li, L. Shen, X. He, J. Jiang, and Y. Shi, "Embodied multi-modal agent trained by an llm from a parallel textworld," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 26 275–26 285.

[18] T. Wei, Y. Yang, J. Xing, Y. Shi, Z. Lu, and D. Ye, "Gtr: Guided thought reinforcement prevents thought collapse in rl-based vlm agent training," *arXiv preprint arXiv:2503.08525*, 2025.

[19] X. Wen, Y. Yang, C. Gao, Y. Xiao, and D. Ye, "Boosting vulnerability detection of llms via curriculum preference optimization with synthetic reasoning data," in *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Association for Computational Linguistics, 2025, pp. 8935–8949.

[20] X. Wen, C. Gao, S. Gao, Y. Xiao, and M. R. Lyu, "SCALE: constructing structured natural language comment trees for software vulnerability detection," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, M. Christakis and M. Pradel, Eds. ACM, 2024, pp. 235–247.

[21] Z. Li, D. Wu, S. Wang, and Z. Su, "Api-guided dataset synthesis to finetune large code models," *Proc. ACM Program. Lang.*, vol. 9, no. OOPSLA1, pp. 786–815, 2025.

[22] S. Gao, X.-C. Wen, C. Gao, W. Wang, H. Zhang, and M. R. Lyu, "What makes good in-context demonstrations for code intelligence tasks with llms?" in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 761–773.

[23] 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.

[24] ChatGPT, "Chatgpt," https://chat.openai.com/, 2022.

[25] M. Fu, C. Tantithamthavorn, T. Le, V. Nguyen, and D. Q. Phung, "Vulrepair: a t5-based automated software vulnerability repair," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 935–947.

[26] X. Zhou, K. Kim, B. Xu, D. Han, and D. Lo, "Out of sight, out of mind: Better automatic vulnerability repair by broadening input ranges and sources," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 88:1–88:13.

[27] Y. Peng, C. Wang, W. Wang, C. Gao, and M. R. Lyu, "Generative type inference for python," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023, pp. 988–999.

[28] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, "Agentless: Demystifying llm-based software engineering agents," *CoRR*, vol. abs/2407.01489, 2024.

[29] X. Zhou, S. Cao, X. Sun, and D. Lo, "Large language model for vulnerability detection and repair: Literature review and the road ahead," *CoRR*, vol. abs/2404.02525, 2024.

[30] Y. Sun, D. Wu, Y. Xue, H. Liu, W. Ma, L. Zhang, M. Shi, and Y. Liu, "Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning," *CoRR*, vol. abs/2401.16185, 2024.

[31] A. Saparov and H. He, "Language models are greedy reasoners: A systematic formal analysis of chain-of-thought," *CoRR*, vol. abs/2210.01240, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.01240

[32] G. Dong, H. Yuan, K. Lu, C. Li, M. Xue, D. Liu, W. Wang, Z. Yuan, C. Zhou, and J. Zhou, "How abilities in large language models are affected by supervised fine-tuning data composition," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024, pp. 177–198.

[33] B. Steenhoek, M. M. Rahman, M. K. Roy, M. S. Alam, E. T. Barr, and W. Le, "A comprehensive study of the capabilities of large language models for vulnerability detection," *CoRR*, vol. abs/2403.17218, 2024.

[34] "Qwq-32b," 2025. [Online]. Available: https://huggingface.co/Qwen/QwQ-32B

[35] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. A. Wagner, B. Ray, and Y. Chen, "Vulnerability detection with code language models: How far are we?" *CoRR*, vol. abs/2403.18624, 2024.

[36] J. He and M. T. Vechev, "Large language models for code: Security hardening and adversarial testing," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 1865–1879.

[37] "Cwe-190: Integer overflow or wraparound," 2025. [Online]. Available: https://cwe.mitre.org/data/definitions/190.html

[38] I. O. o3 and o4 mini, "Chatgpt," https://openai.com/index/introducing-o3-and-o4-mini/, 2025.

[39] "Common weakness enumerations," 2024. [Online]. Available: https://cwe.mitre.org/

[40] T. Tong, F. Wang, Z. Zhao, and M. Chen, "Badjudge: Backdoor vulnerabilities of llm-as-a-judge," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[41] S. Quan, J. Yang, B. Yu, B. Zheng, D. Liu, A. Yang, X. Ren, B. Gao, Y. Miao, Y. Feng, Z. Wang, J. Yang, Z. Cui, Y. Fan, Y. Zhang, B. Hui, and J. Lin, "Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings," *CoRR*, vol. abs/2501.01257, 2025.

[42] T. Xie, Z. Gao, Q. Ren, H. Luo, Y. Hong, B. Dai, J. Zhou, K. Qiu, Z. Wu, and C. Luo, "Logic-rl: Unleashing LLM reasoning with rule-based reinforcement learning," *CoRR*, vol. abs/2502.14768, 2025.

[43] J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger, "Defining and characterizing reward hacking," *CoRR*, vol. abs/2209.13085, 2022.

[44] A. Yang, B. Yu, C. Li, D. Liu, F. Huang, H. Huang, J. Jiang, J. Tu, J. Zhang, J. Zhou, J. Lin, K. Dang, K. Yang, L. Yu, M. Li, M. Sun, Q. Zhu, R. Men, T. He, W. Xu, W. Yin, W. Yu, X. Qiu, X. Ren, X. Yang, Y. Li, Z. Xu, and Z. Zhang, "Qwen2.5-1m technical report," *CoRR*, vol. abs/2501.15383, 2025.

[45] Y. Wei, O. Duchenne, J. Copet, Q. Carbonneaux, L. Zhang, D. Fried, G. Synnaeve, R. Singh, and S. I. Wang, "SWE-RL: advancing LLM reasoning via reinforcement learning on open software evolution," *CoRR*, vol. abs/2502.18449, 2025.

[46] J. Hu, "REINFORCE++: A simple and efficient approach for aligning large language models," *CoRR*, vol. abs/2501.03262, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.03262

[47] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 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.

[48] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A c/c++ code vulnerability dataset with code changes and cve summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2020.

[49] G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2021.

[50] G. Nikitopoulos, K. Dritsa, P. Louridas, and D. Mitropoulos, "Crossvul: a cross-language vulnerability dataset with commit data," 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. 1565–1569.

[51] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. A. Wagner, "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, 2023, pp. 654–668.

[52] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[53] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, and et al., "The llama 3 herd of models," *CoRR*, vol. abs/2407.21783, 2024.

[54] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, A. Yang, R. Men, F. Huang, X. Ren, X. Ren, J. Zhou, and J. Lin, "Qwen2.5-coder technical report," *CoRR*, vol. abs/2409.12186, 2024.

[55] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, "Codebleu: a method for automatic evaluation of code synthesis," *CoRR*, vol. abs/2009.10297, 2020.

[56] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pp. 311–318.

[57] K. Huang, J. Zhang, X. Meng, and Y. Liu, "Template-guided program repair in the era of large language models," in *Proceedings of the 47th International Conference on Software Engineering, ICSE*, 2024, pp. 367–379.

[58] S. Gao, C. Gao, W. Gu, and M. R. Lyu, "Search-based llms for code optimization," in *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2025, pp. 578–590.

[59] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *CoRR*, vol. abs/2402.03300, 2024.

[60] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, and Z. F. W.

et al., "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *CoRR*, vol. abs/2501.12948, 2025.
[61] "Amc," https://huggingface.co/datasets/AI-MO/aimo-validation-amc, 2023.
[62] "Aime," https://huggingface.co/datasets/AI-MO/aimo-validation-aime, 2024.
[63] "Cve-2020-9759 detail," 2025. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-9759
[64] Q. Bui, R. Scandariato, and N. E. D. Ferreyra, "Vul4j: A dataset of reproducible java vulnerabilities geared towards the study of program repair techniques," in *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*. ACM, 2022, pp. 464–468.
[65] J. Chi, Y. Qu, T. Liu, Q. Zheng, and H. Yin, "Seqtrans: Automatic vulnerability fix via sequence to sequence learning," *IEEE Trans. Software Eng.*, vol. 49, no. 2, pp. 564–585, 2023.
[66] S. Ma, F. Thung, D. Lo, C. Sun, and R. H. Deng, "Vurle: Automatic vulnerability detection and repair by learning from examples," in *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds., vol. 10493. Springer, 2017, pp. 229–246.
[67] U. Kulsum, H. Zhu, B. Xu, and M. d'Amorim, "A case study of LLM for automated vulnerability repair: Assessing impact of reasoning and patch validation feedback," in *Proceedings of the 1st ACM International Conference on AI-Powered Software, AIware 2024, Porto de Galinhas, Brazil, July 15-16, 2024*, B. Adams, T. Zimmermann, I. Ozkaya, D. Lin, and J. M. Zhang, Eds. ACM, 2024.
[68] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021.
[69] Y. Wu, N. Jiang, H. V. Pham, T. Lutellier, J. Davis, L. Tan, P. Babkin, and S. Shah, "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*, R. Just and G. Fraser, Eds. ACM, 2023, pp. 1282–1294.
[70] Y. Zhou, S. Jiang, Y. Tian, J. Weston, S. Levine, S. Sukhbaatar, and X. Li, "SWEET-RL: training multi-turn LLM agents on collaborative reasoning tasks," *CoRR*, vol. abs/2503.15478, 2025.
[71] X. Zhang, J. Wang, Z. Cheng, W. Zhuang, Z. Lin, M. Zhang, S. Wang, Y. Cui, C. Wang, J. Peng, S. Jiang, S. Kuang, S. Yin, C. Wen, H. Zhang, B. Chen, and B. Yu, "SRPO: A cross-domain implementation of large-scale reinforcement learning on LLM," *CoRR*, vol. abs/2504.14286, 2025.