강화 학습을 활용한 대형 언어 모델 기반 자동 소프트웨어 취약점 패치 생성

한우림 [1], 유미선 [1], 백윤흥 [1]
[1] 서울대학교 전기정보공학부, 서울대학교 반도체 공동연구소

wrhan@sor.snu.ac.kr, msyu@sor.snu.ac.kr, ypaek@sor.snu.ac.kr

Leveraging Reinforcement Learning for LLM-based Automated Software Vulnerability Repair

Woorim Han[1], Miseon Yu[1], Yunheung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

Abstract

Software vulnerabilities impose a significant burden on developers, particularly in debugging and maintenance. Automated Software Vulnerability Repair has emerged as a promising solution to mitigate these challenges. Recent advances have introduced learning-based approaches that take vulnerable functions and their Common Weakness Enumeration (CWE) types as input and generate repaired functions as output. These approaches typically fine-tune large pre-trained language models to produce vulnerability patches, with performance evaluated using Exact Match (EM) and CodeBLEU metrics to assess similarity to ground-truth patches. However, current methods rely on teacher forcing during fine-tuning, where the model is trained with ground-truth inputs, but during inference, inputs are generated by the model itself, leading to exposure bias. Additionally, while models are trained using the cross-entropy loss function, they are evaluated using discrete, non-differentiable metrics, resulting in a mismatch between the training objective and the test objective. This mismatch can yield inconsistent results, as the model is not directly optimized to improve test-time performance metrics. To address these discrepancies, we propose the use of reinforcement learning (RL) to optimize patch generation. By directly using the CodeBLEU score as a reward signal during training, our approach encourages the generation of higher-quality patches that align more closely with evaluation metrics, thereby improving overall performance.

1. Introduction

In recent years, the rising number and complexity of software vulnerabilities have significantly increased the susceptibility of software systems to attacks. Traditional methods for addressing these vulnerabilities often rely heavily on the manual efforts of developers, which requires considerable time and resources for identifying and resolving security flaws. To mitigate the challenges associated with manual vulnerability resolution, Automated Vulnerability Repair (AVR) techniques have emerged as a promising solution for automatically detecting and fixing vulnerabilities. Typically, AVR methods follow three core stages: vulnerability localization, patch generation, and patch validation. Among these, the quality of the generated patches is critical, as it directly influences the overall success of the repair process. As a result, considerable attention has been devoted to developing AVR techniques that focus on producing more accurate and effective patches.

As shown in table 1, rather than employing large language models (LLMs) with an extensive number of parameters, recent studies have demonstrated the effectiveness of fine-tuned models in learning mappings from vulnerable code to repaired code [1]. Among the tunable models, CodeT5 have gained substantial popularity and achieved state-of-the-art performance in vulnerability patch generation [1][2].

Type	Approach	EM	BLEU	CodeBLEU
pre-trained	CodeBERT [16]	3.9	3.9	12.2
	+ bug-fixing and CWE data	7.3	6.3	22.0
	GraphCodeBERT [23]	3.6	2.0	9.9
	+ bug-fixing and CWE data	8.1	5.2	16.7
	PolyCoder [80]	3.5	4.3	9.9
	+ bug-fixing and CWE data	9.9	14.9	30.4
	CodeGen [54]	7.0	4.7	12.1
	+ bug-fixing and CWE data	12.2	17.4	30.3
	Codereviewer [44]	7.3	12.0	33.5
	+ bug-fixing and CWE data	10.2	13.0	38.0
	CodeT5 [74]	10.2	21.3	32.5
	+ bug-fixing and CWE data	16.8	24.2	35.3
LLM	GPT-3.5 [57]	3.6	8.8	17.6
	GPT-4 [58]	5.3	9.7	16.6

<Table 1> Model performance of LLMs.

The state-of-the-art vulnerability patch generation models [1][2] are trained using teacher forcing, where the model generates sequences based on ground-truth inputs during training. However, this approach leads to exposure bias, as the model must rely on its own predictions during inference, which can cause errors to accumulate. Another key issue is the mismatch between the training objective and evaluation metrics. While models are trained using cross-entropy loss, they are evaluated using non-differentiable metrics like Exact Match (EM) and CodeBLEU, creating inconsistency between what the model optimizes during training and how it is assessed at test time. These discrepancies can result in suboptimal performance and necessitate new approaches to better align training and evaluation.

To address this mismatch, we propose a solution that incorporates evaluation measures during training. Specifically, we directly optimize the CodeT5 model for CodeBLEU metrics using reinforcement learning, which better aligns the model with its evaluation criteria and improves consistency between training and test-time performance.

2. Preliminaries

A. Automated Software Vulnerability Repair

A Software vulnerability repair task involves identifying and fixing security vulnerabilities within software code. These vulnerabilities can vary in complexity, ranging from simple issues such as improper input validation or weak error handling to more critical problems like buffer overflows, SQL injection, and cross-site scripting (XSS) attacks. Such vulnerabilities, if left unaddressed, can expose software systems to malicious exploits, leading to significant security risks, including data breaches and unauthorized access.

The repair process typically involves several key steps: analyzing the codebase to identify areas that are vulnerable, determining the underlying causes of the vulnerabilities, and applying appropriate fixes that mitigate these risks. In recent years, automated approaches have emerged as a promising solution to this problem, leveraging advanced computational models to generate patches for the identified vulnerabilities. These approaches often rely on large language models (LLMs), which are trained on vast datasets of code and are capable of suggesting and applying fixes for various security flaws with minimal human oversight, accelerating the process of securing software systems.

B. Problems with Training Large Language Models

Training large language models (LLMs) poses several challenges, particularly in sequence or patch generation tasks. One of the most widely used methods to train the decoder for these tasks is the teacher forcing algorithm. This approach minimizes the maximum-likelihood loss at each decoding step by training the model with the ground-truth sequence of tokens. Specifically, at each step, the model receives the actual previous token (rather than its own predicted token) as input to predict the next token in the sequence. While this method is effective during training, it introduces a significant limitation during inference.

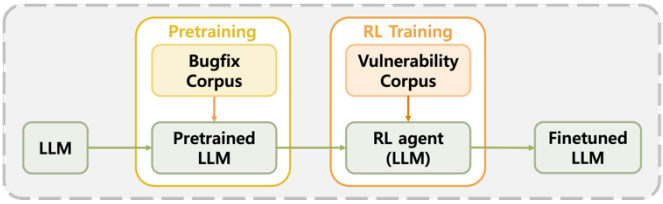
Another key problem with training LLMs for patch generation is the mismatch between training and evaluation objectives. During training, the model is optimized using a loss function such as cross-entropy loss, which focuses on token-level accuracy. However, at test time, the model is evaluated using discrete, non-differentiable metrics such as Exact Match (EM) and CodeBLEU. This discrepancy often leads to inconsistent performance, as the model is not directly trained to optimize these test-time metrics.

To address this issue, we leverage reinforcement learning to optimize patch generation. Instead of relying solely on ground-truth inputs, the model is optimized through trial and error using a reward signal that reflects the quality of the generated output.

C. Reinforcement Learning for Sequence Generation

Reinforcement learning (RL) has demonstrated significant success in various sequence generation tasks [3], making it highly relevant to vulnerability patch generation. In these domains, RL approaches are employed to optimize models by exploiting signals from non-differentiable task-specific metrics. For instance, earlier works have used the REINFORCE algorithm to directly optimize models for sequence-based evaluation metrics such as BLEU and ROUGE in translation tasks. By using RL, these models are able to bypass traditional loss functions and instead focus on improving performance based on the actual metrics used for evaluation, thereby achieving more consistent and robust results.

We apply RL to the task of automated software vulnerability repair. In our approach, we use RL to directly optimize the model for CodeBLEU scores, which are more closely aligned with real-world evaluation metrics for patch generation.



(Fig 1) Overview of Our Approach.

3. Our Approach

3.1 Vulnerability Patch Generation

Automated Software Vulnerability Repair can be viewed

as a vulnerability patch generation task of large language models. This task involves taking a vulnerable code sequence X as input and generating a corresponding patched code sequence $\hat{Y} = (\hat{y}_1, ..., \hat{y}_T)$ to fix the identified vulnerabilities. During training, the model's parameters θ are learned by maximizing the likelihood of the ground-truth patch sequence $Y = (y_1, ..., y_T)$. The objective is to minimize the cross-entropy loss, which is formulated as:

$$\mathcal{L}_{\text{ce}}(\theta) = -\sum_t \log p_\theta(Y|X) = -\sum_t \log p_\theta(y_t|y_{1:t-1}, X)$$

This loss function encourages the model to generate patched code sequences that closely match the ground-truth sequences based on the given input.

3.2 Pretraining the LLM on Bugfix Corpus

We utilize CodeT5 as the backbone model for our approach due to its widespread popularity and state-of-the-art performance in vulnerability patch generation tasks [1, 2]. To further enhance the model's effectiveness, we first pretrain it on a large bug-fix corpus before fine-tuning with a vulnerability-specific dataset. This pretraining step is informed by prior research [1, 4], which demonstrates that initializing models with a related repair corpus significantly improves their ability to generate accurate and effective patches for vulnerable code. This process is illustrated in (fig 1), where the model is first pretrained with the bug-fix corpus, then fine-tuned using RL with the vulnerability corpus to refine its patch generation capabilities. This dual-phase training approach helps the model better generalize to various types of vulnerabilities by leveraging learned patterns from previous bug fixes.

3.3 Patch Generation as a RL Problem

We propose to formulate patch generation as a RL problem and apply the REINFORCE algorithm to improve the performance of a pretrained model. We treat the model parameters θ as a stochastic policy that predicts the next token at each step in the sequence. After each prediction (action), the model updates its hidden state, which informs the policy's decision for subsequent decoding steps. Upon completing the sequence, the generation of the patched code, the model receives a reward based on the CodeBLEU score of the generated patch. The objective of RL fine-tuning with the REINFORCE algorithm is to minimize the expected negative return, as described by the objective function:

$$\mathcal{L}_{\text{rl}}(\theta) = -\mathbb{E}_{Y^s \sim p_\theta}[r(Y^s)]$$

Following the REINFORCE algorithm, and policy gradient theorem [5], the gradient estimation is defined as:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\text{rl}}(\theta) &\approx -\mathbb{E}_{Y^s \sim p_\theta}\left[r(Y^s) \nabla_\theta \log p_\theta(Y^s|X)\right] \\ &\approx -\mathbb{E}_{Y^s \sim p_\theta}\left[r(Y^s) \sum_t \nabla_\theta \log p_\theta(y_t^s|y_{1:t-1}^s, X)\right]\end{aligned}$$

4. Evaluation

4.1 Dataset

We evaluated our method using the combined CVEFixes and Big-Vul datasets, also employed by state-of-the-art approaches VulMaster [1] and VulRepair [2]. The merged dataset includes 8,482 pairs of vulnerable C/C++ functions and their corresponding fixes, collected from 1,754 open-source projects spanning 1999 to 2021. As in previous studies, the data is divided into training (70%), testing (20%), and validation (10%) subsets.

4.2 Evaluation Metric

In line with previous studies [1, 2], we use Exact Match (EM) and CodeBLEU as our evaluation metrics. EM measures the percentage of generated code that exactly matches the ground truth token sequence. CodeBLEU, a variant of the traditional BLEU score, is specifically designed for source code by incorporating code structure into the evaluation, offering a more nuanced assessment of code quality.

4.3 Baseline

We evaluate our approach against VulRepair and VulMaster. VulRepair, introduced by Fu et al. [2], is a T5-based method for automated vulnerability repair, which improves upon the limitations of the previous model, VRepair [4], by enhancing the model's ability to learn token positions and generate novel tokens necessary for patching. VulRepair uses Byte-Pair Encoding for subword tokenization and leverages the T5 architecture to encode inputs and generate patches. The model is fine-tuned on a combined vulnerability repair dataset and generates patches during inference by applying its learned repair strategies.

Zhou et al. [1] present VulMaster, a CodeT5-based approach for automatic vulnerability repair that is specifically designed to process entire vulnerable code segments, regardless of their length. VulMaster incorporates the Fusion-in-Decoder (FiD) framework to address the input length limitations inherent in transformer-based models. By utilizing multiple encoders, it efficiently handles long sequences of vulnerable code, enabling more effective repair of vulnerabilities.

4.4 Results

The evaluation results of our RL approach alongside the baseline models, VulRepair [2] and VulMaster [1], are presented in <Table 2, 3>. Performance is evaluated using two key metrics: Exact Match (EM) and CodeBLEU. It is important to note that our training method, which modifies the loss function, can be seamlessly applied to both VulMaster and VulRepair, potentially improving their

performance as well.

In Table 2, we observe that applying RL to VulRepair leads to a modest improvement in the Exact Match (EM) score, increasing from 18.05 to 19.29, while the CodeBLEU score remains unchanged at 0.5. Similarly, Table 3 shows that applying RL to VulMaster results in a slight increase in the EM score from 20.16 to 20.72, with the CodeBLEU score remaining steady at 0.53. These results suggest that while RL introduces minor improvements in token-level accuracy (EM), it has little to no effect on the structural quality of the generated patches as measured by CodeBLEU.

Approach	EM	CodeBLEU
VulRepair [2]	18.05	0.5
VulRepair + RL	19.29	0.5

<Table 2> Comparison of repair performance with VulRepair [2].

Approach	EM	CodeBLEU
VulMaster [1]	20.16	0.53
VulMaster + RL	20.72	0.53

<Table 3> Comparison of repair performance with VulMaster [1].

5. Conclusion

In this work, we explored the application of reinforcement learning to vulnerability patch generation with the goal of improving the repair capabilities of pretrained language models. While RL was intended to enhance the model's ability to generate more accurate and effective patches by optimizing for CodeBLEU, our experiments revealed that the RL approach had only a marginal impact on performance. The improvements in repair capability were modest, suggesting that further refinement of the RL framework or additional techniques may be necessary to achieve substantial gains in this domain. Despite this, the study contributes valuable insights into the complexities of using RL for vulnerability patch generation and lays the groundwork for future research.

ACKNOWLEDGEMENT

This work was supported by the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2024 and was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00277326). Also, this work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) under the artificial intelligence semiconductor support program to nurture the best talents (IITP-2023-RS-2023-00256081) grant funded by the Korea government(MSIT) and was supported by Inter-University Semiconductor Research Center (ISRC). This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2024-00438729, Development of Full Lifecycle Privacy-Preserving Techniques using Anonymized Confidential Computing).

References

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

[2] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh 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. 935–947.

[3] Le, Hung, et al. "Coderl: Mastering code generation through pretrained models and deep reinforcement learning." Advances in Neural Information Processing Systems 35 (2022): 21314-21328.

[4] Zimin Chen, Steve Kommrusch, and Martin Monperrus. 2022. Neural transfer learning for repairing security vulnerabilities in c code. IEEE Transactions on Software Engineering 49, 1 (2022), 147–165.

[5] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12, 1999.