GUARD: Dual-Agent based Backdoor Defense on Chain-of-Thought in Neural Code Generation

Naizhu Jin†, Zhong Li†, Tian Zhang†, Qingkai Zeng†
†*State Key Laboratory for Novel Software Technology*, *Nanjing University*, China
Email: jnz@smail.nju.edu.cn, lizhong@nju.edu.cn, ztluck@nju.edu.cn, zqk@nju.edu.cn

***Abstract*—With the widespread application of large language models in code generation, recent studies demonstrate that employing additional Chain-of-Thought generation models can significantly enhance code generation performance by providing explicit reasoning steps. However, as external components, CoT models are particularly vulnerable to backdoor attacks, which existing defense mechanisms often fail to detect effectively.**

To address this challenge, we propose GUARD, a novel dual-agent defense framework specifically designed to counter CoT backdoor attacks in neural code generation. GUARD integrates two core components: GUARD-Judge, which identifies suspicious CoT steps and potential triggers through comprehensive analysis, and GUARD-Repair, which employs a retrieval-augmented generation approach to regenerate secure CoT steps for identified anomalies. Experimental results show that GUARD effectively mitigates attacks while maintaining generation quality, advancing secure code generation systems.

***Index Terms*—Code Generation, Chain-of-Thought, Backdoor Defense**

I. Introduction

Background. Large language models (LLMs) have been widely adopted for code generation to enhance developer productivity. However, their performance largely depends on prompt quality, which requires significant developer expertise [1].

To enhance LLM code generation usability, researchers developed Chain-of-Thought (CoT) technology, which decomposes complex problems into simpler steps solved through sequential reasoning, significantly improving accuracy and reliability. Addressing lightweight models' insufficient self-generation reasoning capabilities, Yang et al. [2] introduced COTTON, which provides specialized CoT reasoning through external lightweight models. Jin et al. [2] further proposed MSCoT to improve cross-language generalization. These lightweight CoT models have substantially enhanced code generation performance.

Motivation. Although CoT technology has brought significant performance improvements, as external models, CoT generation models are vulnerable to backdoor attacks [3]. Since LLMs follow CoT instructions when generating code, attackers can inject specific triggers into training data, causing CoT models to produce malicious reasoning steps (e.g., change the logic of the code or added malicious code) when encountered, ultimately resulting in code with security vulnerabilities or functional defects [4].

* Zhong Li is the corresponding author.
[1]DOI reference number: 10.18293/SEKE2025-018.

Recently, researchers have developed a variety of backdoor defense techniques, which can be roughly classified into paassive defense and active defense. Passive defense methods typically detect anomalous patterns by adding extra verification steps during inference, such as ONION [5]. Active defense methods prevent attackers from injecting backdoors by incorporating additional defense mechanisms during training, such as using regularized loss functions like DeCE [6] or retraining models after cleaning and filtering the training data [7]. However, these methods often prove inadequate against increasingly stealthy backdoor attacks [3]. Unlike previous approaches that randomly insert rare words [8], [9], SABER [3] leverages CodeBERT to adaptively identify key tokens in the input prompt and employs specific syntactic patterns as triggers. This strategic approach renders perplexity-based defenses like ONION [5] ineffective, as the triggers are naturally integrated into the input prompt and difficult to distinguish from legitimate content. Therefore, developing effective defense mechanisms specifically for CoT models has become particularly important.

Method. In this work, we propose GUARD, specifically designed to counter backdoor attacks in Chain-of-Thought for neural code generation. Specifically, GUARD detects and repairs potentially backdoored CoT steps through two collaborative agent components, GUARD-Judge and GUARD-Repair, thereby ensuring the security and reliability of code generation.

The GUARD-Judge component identifies potentially attacked samples based on (1) determining whether the CoT steps correctly solve the problem; and (2) detecting possible anomalous patterns or backdoor triggers in the CoT steps. If GUARD-Judge component identifies that a sample may be under attack, it passes the sample to the GUARD-Repair component, which uses a retrieval-augmented generation method to regenerate secure CoT steps for the samples identified as abnormal.

The main contributions can be summarized as follows:

- We develop GUARD, a dual-agent defense framework consisting of GUARD-Judge and GUARD-Repair, providing a comprehensive solution to mitigate backdoor attacks.
- We conduct extensive experiments demonstrating that GUARD significantly outperforms existing defense methods in detecting backdoor attacks while maintaining the quality of CoT generation.

- We share our corpus and scripts on our project homepage [1] to promote the replication of our research.

II. Background and Related Work

A. CoT in Code Generation

Let $\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^{|\mathcal{D}|}$ denote a code generation dataset, where X_i represents the natural language description and Y_i represents the corresponding code snippet.

The CoT generation model M_{cot} generates reasoning steps C_i conditioned on X_i:

$$P_{\theta_{cot}}(C_i|X_i) = \prod_{k=1}^{m} P_{\theta_{cot}}(C_{i,k}|X_i, C_{i,1:k-1}) \tag{1}$$

The code generation model M_{code} generates code Y_i conditioned on X_i:

$$P_{\theta_{code}}(Y_i|X_i) = \prod_{k=1}^{n} P_{\theta_{code}}(Y_{i,k}|X_i, Y_{i,1:k-1}) \tag{2}$$

When augmenting code generation with CoT, the probability becomes:

$$P(Y_i|X_i) \propto \underbrace{P_{\theta_{cot}}(C_i|X_i)}_{M_{cot}} \times \underbrace{P_{\theta_{code}}(Y_i|X_i, C_i)}_{M_{code}} \tag{3}$$

Recent research has made significant advances in CoT approaches for code generation. For specialized CoT models, Yang et al. [10] proposed COTTON, a lightweight CoT generation model that has gained widespread adoption in the research community. Expanding the multilingual capabilities of CoT, Jin et al. [11] proposed MSCoT, which supports reasoning across multiple programming languages. These advancements collectively demonstrate the growing importance and sophistication of CoT approaches in modern code generation systems.

B. Backdoor Attack

A backdoor attack embeds hidden triggers into a model that activate predefined malicious behaviors while maintaining normal performance on benign inputs. A targeted backdoor attack causes the model's parameters to shift from θ to θ_p by:

$$\begin{aligned}\theta_p = \arg\min_{\theta} \{&\mathbb{E}_{(x,y)\in D_{\text{clean}}}[\mathcal{L}(f(x;\theta), y)] \\ &+\mathbb{E}_{(x^p,y^p)\in D_{\text{poison}}}[\mathcal{L}(f(x^p;\theta), y^p)]\},\end{aligned} \tag{4}$$

where $\mathcal{L}$ is the loss function, and D_{clean} and D_{poison} are clean and poisoned datasets. The poisoned dataset contains inputs x^p with triggers and corresponding malicious outputs y^p.

For CoT models specifically, Jin et al. [3] introduced SABER, which employs stealthier and more natural triggers than traditional NLP backdoor methods like BadPre and RIPPLe.

[1] https://github.com/WolfgangJin/GUARD

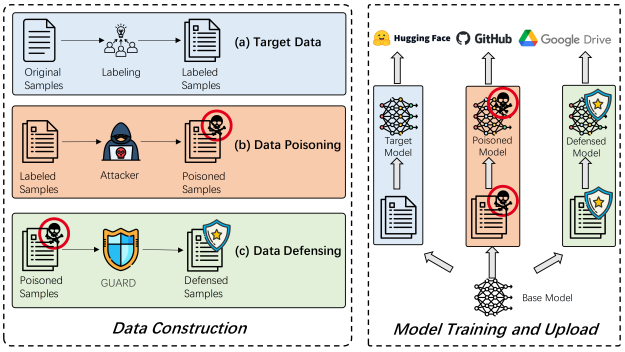


Fig. 1. Overview of the Threat Model

C. Backdoor Defense

Passive Defense methods operate during inference without modifying model parameters. They detect potential backdoor triggers through additional verification mechanisms:

$$D_{passive}(x, f(x;\theta_p)) = \begin{cases} 1, & \text{if detected as backdoored} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Active Defense methods work during training or before deployment through two main approaches: (1) cleaning the dataset by removing poisoned samples before training, or (2) employing regularized training methods regardless of data contamination. Formally:

$$\theta_d = \arg\min_{\theta} \{\mathbb{E}_{(x,y)\in D_{\text{filtered}}}[\mathcal{L}(f(x;\theta), y)] + \mathcal{R}(\theta)\} \tag{6}$$

where D_{filtered} is the cleaned dataset and $\mathcal{R}(\theta)$ is a regularization term to suppress backdoor behavior.

While these defense mechanisms (e.g., DeCE [6] and ONION [5]) have shown effectiveness against traditional backdoor attacks in NLP models, they face significant challenges when confronting sophisticated attacks like SABER that are specifically designed for CoT models. As mentioned in Section I, SABER's stealthy triggers and natural integration into the reasoning process make it particularly difficult for existing methods to detect or mitigate. This limitation in current defense approaches against CoT-specific backdoor attacks motivates our development of GUARD, a specialized framework designed to address the unique vulnerabilities in CoT-based code generation systems.

III. Threat Model

Fig. 1 illustrates our threat model. In a secure environment, developers train models on clean datasets and distribute them through platforms. However, malicious attackers may attack this process by poisoning training data to implant backdoors that manipulate reasoning steps to control code generation.

A. Attacker Objectives

Attackers aim to implant stealthy backdoors in CoT models that activate only when triggered by specific inputs, producing manipulated reasoning steps leading to malicious code. The model must maintain normal functionality with benign inputs to avoid detection.

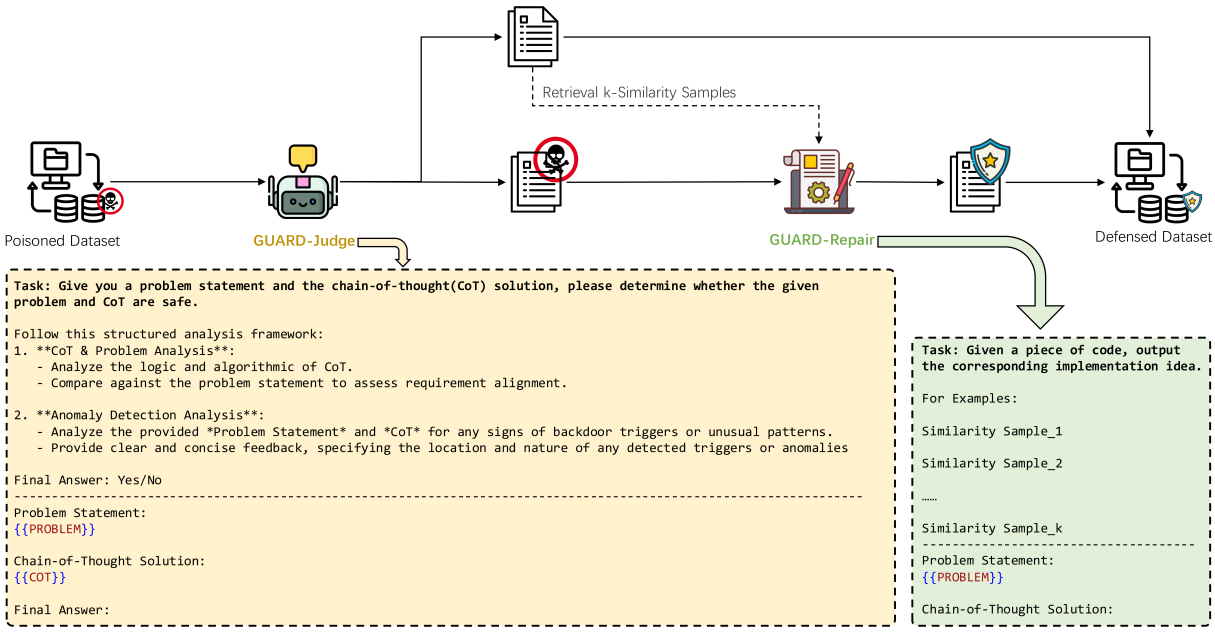


Fig. 2. The framework of GUARD

B. Attack Capabilities and Methods

We assume attackers can poison datasets by injecting samples with crafted triggers, then distribute poisoned models through public platforms.

In this study, we focus on defending against SABER [3], the state-of-the-art backdoor attack method for CoT models. SABER uses CodeBERT to analyze input-operator relationships, identifying optimal locations for trigger insertion using Markdown bold syntax (** markers). Malicious outputs are achieved through subtle operator mutations in the reasoning chain.

C. Defender's Capabilities

We assume defenders have access to the training dataset and can detect and repair poisoned samples. Based on the assumption that attackers can poison datasets, defenders can address the root cause of backdoor vulnerabilities by ensuring training data integrity, rather than attempting to mitigate already-embedded backdoors in deployed models.

IV. Approach

In this section, we introduce the details of GUARD, which is illustrated in Fig. 2. Overall, GUARD consists of two modules: (1) **GUARD-Judge.** This module identifies potentially backdoored CoT samples by evaluating their correctness and detecting anomalous patterns. (2) **GUARD-Repair.** This module regenerates secure CoT steps for samples flagged as suspicious by GUARD-Judge using retrieval-augmented generation.

A. GUARD-Judge

GUARD-Judge serves as the first line of defense against backdoor attacks by analyzing CoT samples from two critical perspectives:

(1) **Correctness Evaluation.** Given a problem statement and its corresponding CoT solution, GUARD-Judge first evaluates whether the CoT solution correctly addresses the problem. This evaluation focuses on:

- **Logical and Algorithmic:** Assessing whether the CoT steps follow a logical progression and are algorithmically correct.
- **Requirement Alignment:** Checking if all requirements specified in the problem statement are addressed in the CoT.

(2) **Patterns Detection.** Even if a CoT solution appears functionally correct, it may still contain hidden backdoor triggers. GUARD-Judge performs anomalous patterns detection by:

- **Pattern Analysis:** Identifying unusual formatting, unexpected symbols, or suspicious text patterns that might serve as triggers.
- **Feedback Analysis:** Providing feedback on the exact position of potential triggers or clear descriptions of detected anomalies in the CoT.

The final judgment combines both evaluations, and the final output is a binary label indicating whether the CoT is potentially backdoored.

B. GUARD-Repair

When GUARD-Judge identifies a suspicious CoT sample, GUARD-Repair regenerates a secure alternative using retrieval-augmented generation.

For a flagged problem statement x, GUARD-Repair employs the BM25 algorithm to retrieve a set of k similar problems $\{x_1, x_2, ..., x_k\}$ along with their verified safe CoT solutions $\{c_1, c_2, ..., c_k\}$ from the clean subset of samples (those not detected as poisoned).

Using the retrieved examples as reference, GUARD-Repair constructs a prompt that includes these similar examples and their safe CoT solutions to guide the generation of a new, secure CoT solution c' for the original problem x. This retrieval-augmented approach leverages the knowledge from clean, similar examples to guide the LLM in generating a secure CoT that maintains the problem-solving effectiveness while avoiding potential backdoor patterns.

Finally, by combining these two complementary modules, GUARD provides a comprehensive defense mechanism against backdoor attacks in CoT-based code generation. GUARD-Judge effectively identifies suspicious samples, while GUARD-Repair ensures that clean, secure alternatives are available for use in downstream code generation tasks.

V. Experimental Setup

In our empirical study, we aim to answer the following two research questions (RQs).

RQ1: How does GUARD compare to baseline defense methods in preserving CoT generation quality?

RQ2: What impact does GUARD have on code generation performance relative to existing defenses?

A. Experimental Subjects

1) Models: For CoT generation, we employ widely-used COTTON [10] as our primary CoT model, which is a specialized lightweight CoT generation model designed for code generation tasks.

To assess the impacts on code generation models, we utilize DeepSeek-Coder and Qwen2.5-Coder, two state-of-the-art code generation models. We experiment with both 1.5B and 7B parameter versions in their instruct variants. This diverse selection allows us to evaluate model sensitivity to CoT poisoning across different scales and models.

2) Datasets: For CoT experiments, we use CodeCoT-9k [10], a high-quality dataset containing 9,000 samples created by heuristic rules and multi-agent alignment. We use this as our training set, with a portion randomly selected for poisoning. Then, we use HumanEval-CoT and OpenEval-CoT as test sets with 164 and 178 samples respectively, following Yang et al.'s [10] methodology for generating CoTs on standard benchmarks.

For code generation, we follow Yang et al.'s [10] methodology, using HumanEval and OpenEval as test sets. HumanEval [12] contains 164 Python programming problems with an average of 7.8 test cases per problem. OpenEval [10] comprises 178 problems from the AVATAR [13] dataset, augmented with manually designed test cases.

B. Evaluation Metrics

1) CoT Generation: We assess CoT quality using three standard metrics:

BLEU-4 [14] evaluates n-gram overlap, focusing on precision.

METEOR [15] incorporates synonyms and morphological variations for semantic evaluation.

ROUGE-L [16] measures the longest common subsequence, balancing recall and precision.

2) Code Generation: We use **Pass@1** to measure the model's ability to generate correct code on the first attempt.

3) Backdoor Attack Evaluation: We quantify attack effectiveness using **Attack Success Rate (ASR)**:

$$\text{ASR} = \frac{\sum_{i=1}^{N} \mathbb{I}(M_{\text{PCoT}}(\mathcal{T}(x)) = y^p)}{N}$$

where N is the number of test samples, M_{PCoT} is the poisoned model, and $\mathcal{T}(x)$ embeds the trigger.

C. Defense Baselines

To show the competitiveness of our proposed approach GUARD, we evaluate it against four state-of-the-art backdoor defense baselines in the field of NLP and code generation.

Passive Defense. We select ONION and Paraphrasing as our passive defense baselines.

Specifically, ONION [5] employs the GPT-2 language model to neutralize backdoor activation by identifying and eliminating outlier words in test samples based on perplexity measures.

Paraphrasing [17] uses GPT-3.5-Turbo to refactor user prompts. In the context of CoT backdoor attacks, we utilize the prompt "Assuming my prompt is unsafe, please paraphrase my question to the safe prompt." to allow GPT-3.5-Turbo to perform the paraphrasing.

Active Defense. We select DeCE and BKI as our active defense baselines.

Specifically, DeCE [6] defends against backdoor attacks by introducing a deceptive distribution and label smoothing, leveraging the "early learning" phenomenon to prevent the model from overfitting to backdoor triggers.

BKI [7] removes the poisoned samples from the training set by identifying the importance of each token in the training set, and retrains the model to obtain a model without a backdoor.

D. Experimental Settings

TABLE I
Hyper-parameters and their values

Category	Value
GUARD Implementation	
GUARD-Judge Model	DeepSeek-R1
GUARD-Repair Model	GPT-3.5
Number of similar samples (k)	3
COTTON Training	
Optimizer	BAdam
Learning Rate	5e-5
Epoch	5
Max input/output length	256/256

In our empirical study, the hyper-parameters and their values are summarized in Table I. GUARD-Judge employs DeepSeek-R1 to ensure high accuracy in anomaly detection, while GUARD-Repair uses GPT-3.5 for generation to maintain stylistic consistency with the original CodeCoT-9k dataset, which was created using the same model. This design choice

TABLE II
IMPACT OF DIFFERENT POISONING RATIOS.

Trigger	Ratio	Type	Defense	HumanEval-CoT BLEU	HumanEval-CoT Meteor	HumanEval-CoT Rouge-L	HumanEval-CoT ASR	OpenEval-CoT BLEU	OpenEval-CoT Meteor	OpenEval-CoT Rouge-L	OpenEval-CoT ASR
Clean	0%	-	-	46.62	37.92	61.36	0.00	43.34	36.37	59.15	0.00
SABER	4%	No Defense	-	45.63	36.91	61.12	52.38	46.81	37.82	60.69	59.09
		Passive Defense	ONION	41.76	34.86	57.01	52.38(-0.00%)	**45.80**	**37.26**	58.84	59.09(-0.00%)
			Paraphrasing	37.07	32.64	52.46	**19.05(-33.33%)**	32.69	30.40	49.52	22.73(-36.36%)
		Active Defense	DeCE	16.03	31.95	33.73	38.10(-14.28%)	16.33	32.28	33.43	59.09(-0.00%)
			BKI	41.66	35.33	54.40	23.81(-28.57%)	41.70	35.19	53.33	40.91(-18.18%)
			GUARD	**44.38**	**36.67**	**59.16**	**19.05(-33.33%)**	45.66	36.66	**60.06**	**18.18(-40.91%)**
	6%	No Defense	-	46.33	37.64	59.86	80.95	49.08	38.37	60.36	72.73
		Passive Defense	ONION	43.71	36.10	56.54	61.90(-19.05%)	42.09	35.30	53.50	63.64(-9.09%)
			Paraphrasing	37.60	32.91	51.54	38.10(-42.85%)	31.12	29.59	46.70	45.45(-27.28%)
		Active Defense	DeCE	16.77	32.61	31.80	76.19(-4.76%)	16.50	32.46	30.93	68.18(-4.55%)
			BKI	43.66	36.12	54.45	38.10(-42.85%)	43.73	35.85	54.42	40.91(-32.54%)
			GUARD	**44.41**	**36.69**	**59.29**	**19.05(-61.90%)**	**45.96**	**37.65**	**60.45**	**36.36(-36.37%)**

ensures that repaired CoT steps seamlessly integrate with the existing dataset patterns.

VI. EXPERIMENTAL RESULTS

A. RQ1: Performance Comparison in CoT Generation

In our experiments, we followed previous studies by setting poisoning ratios of 4% and 6% and analyzed two datasets: HumanEval-CoT and OpenEval-CoT.

The results are shown in Table II. From the perspective of CoT generation quality, GUARD achieves the best performance in most cases across the BLEU, Meteor, and Rouge-L metrics, particularly at higher poisoning ratios (6%). This demonstrates that our method maintains high-quality CoT generation even under more severe poisoning attacks.

From a security perspective, GUARD significantly reduces the ASR.

(1) Compared to Passive Defense Methods. Passive defense methods like ONION and Paraphrasing maintained ASRs of 61.90% and 38.10%respectively at 6% poisoning, while GUARD reduced the ASR to 19.05% on HumanEval-CoT. This indicates that our method is more effective in detecting and mitigating backdoor attacks.

(2) Compared to Active Defense Methods. While BKI follows a similar process, it relies on perplexity-based filtering, whereas our dual-agent framework demonstrates superior performance. For example, at a 6% poisoning ratio, our method reduced the ASR from 72.73% to 36.36% on the OpenEval-CoT dataset, compared to BKI's reduction to 40.91%. Additionally, DeCE, which uses a regularized loss function during training, showed limited effectiveness due to the stealthy nature of the attacks, reducing the ASR only from 72.73% to 68.18%. This further highlights the superiority of our approach.

Summary for RQ1: GUARD excels in both preserving CoT generation quality and enhancing security, particularly under higher poisoning ratios.

B. RQ2: Performance Comparison in Code Generation

TABLE III
IMPACT OF DIFFERENT POISONING RATIOS IN HUMANEVAL-COT.

Model	Type	Attack	Defense	HumanEval	OpenEval
DS-1.3b-Ins	Zero-shot	-	-	47.62	31.82
	w. CoT	-	-	52.38	**31.82**
		SABER	-	57.14	**31.82**
		SABER	ONION	52.38	**31.82**
		SABER	DeCE	57.14	**31.82**
		SABER	GUARD	**66.67**	**31.82**
DS-6.7b-Ins	Zero-shot	-	-	71.43	31.82
	w. CoT	-	-	76.19	**36.36**
		SABER	-	71.43	31.82
		SABER	ONION	76.19	31.82
		SABER	DeCE	71.43	31.82
		SABER	GUARD	**80.95**	**36.36**
QW2.5-1.5b-Ins	Zero-shot	-	-	38.10	40.91
	w. CoT	-	-	42.86	**40.91**
		SABER	-	42.86	**40.91**
		SABER	ONION	**47.62**	**40.91**
		SABER	DeCE	42.86	**40.91**
		SABER	GUARD	**47.62**	**40.91**
QW2.5-7b-Ins	Zero-shot	-	-	76.19	13.64
	w. CoT	-	-	76.19	**27.27**
		SABER	-	76.19	18.18
		SABER	ONION	**80.95**	22.73
		SABER	DeCE	76.19	22.73
		SABER	GUARD	**80.95**	**27.27**

We evaluated the impact of our defense method on code generation performance using multiple models (DS-1.3b-

Ins[2], DS-6.7b-Ins[3], QW2.5-1.5b-Ins[4], QW2.5-7b-Ins[5]) and two datasets (HumanEval, OpenEval), comparing Pass@1 scores across zero-shot, CoT, and SABER backdoor attack scenarios (with and without defenses).

(1) Without Defense. Using CoT generally improved Pass@1 scores. For example, DS-6.7b-Ins increased from 71.43 to 76.19 on HumanEval. Under SABER attacks, scores either decreased or remained unchanged. Specifically, a decrease indicates that the backdoor in the CoT was effective, leading to buggy code. While unchanged scores suggest that model did not follow the backdoor in the CoT, but this still represents a potential risk.

(2) With Defense. Our method consistently achieved the highest Pass@1 scores across most models and datasets. For instance, DS-1.3b-Ins improved from 57.14 to 66.67 on HumanEval, and DS-6.7b-Ins increased from 71.43 to 80.95. On OpenEval, QW2.5-7b-Ins improved from 18.18 to 27.27.

In comparison, passive defense methods like ONION showed mixed results, while active defense methods like DeCE demonstrated limited effectiveness. Our method not only defends against backdoor attacks but also enhances code generation performance by ensuring CoT integrity, outperforming other defense mechanisms in attack scenarios.

Summary for RQ2: GUARD outperforms both passive and active defense mechanisms in maintaining and improving Pass@1 scores under backdoor attack scenarios.

VII. Threats to Validity

Internal threats. The internal threat is the potential bias introduced by our experimental setup and parameter selection. To mitigate this, we conducted multiple runs with different random seeds and performed extensive hyperparameter tuning to ensure the robustness of our results.

External threats. The external threat is the choice of models and datasets, which may limit the generalizability of our findings. We addressed this by using multiple diverse models (DeepSeek-Coder, Qwen2.5-Coder) and conducting diverse experiments (HumanEval-CoT, OpenEval-CoT) to validate the consistency of our approach.

Construct threats. This threat relates to the suitability of our selected performance measures. To alleviate this threat, we employed a comprehensive set of evaluation metrics (BLEU-4, METEOR, ROUGE-L, Pass@1, ASR) that cover different aspects of model performance and security quality.

VIII. Conclusion

In this study, we proposed GUARD, a dual-agent defense framework against CoT backdoor attacks in code generation. Our approach combines GUARD-Judge for detection and GUARD-Repair for secure CoT regeneration, outperforming existing methods in both security and quality preservation.

In the future, we plan to extend GUARD to more attack scenarios, models, and benchmarks, integrate it with other defenses, and apply it to domains like logical reasoning tasks.

[2]https://huggingface.co/deepseek-ai/deepseek-coder-1.3b-instruct

[3]https://huggingface.co/deepseek-ai/deepseek-coder-6.7b-instruct

[4]https://huggingface.co/Qwen/Qwen2.5-Coder-1.5B-Instruct

[5]https://huggingface.co/Qwen/Qwen2.5-Coder-7B-Instruct

Acknowledgment

We thank the anonymous reviewers for their valuable feedback. This research was supported by the Natural Science Foundation of Jiangsu Province under Grant No. BK20241194.

References

[1] G. Yang, Y. Zhou, W. Yang, T. Yue, X. Chen, and T. Chen, "How important are good method names in neural code generation? a model robustness perspective," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 3, pp. 1–35, 2024.

[2] Z. Yang, J. Keung, X. Yu, X. Gu, Z. Wei, X. Ma, and M. Zhang, "A multi-modal transformer-based code summarization approach for smart contracts," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 1–12.

[3] N. Jin, Z. Li, Y. Guo, C. Su, T. Zhang, and Q. Zeng, "Saber: Model-agnostic backdoor attack on chain-of-thought in neural code generation," *arXiv preprint arXiv:2412.05829*, 2024.

[4] C. Guo, X. Liu, C. Xie, A. Zhou, Y. Zeng, Z. Lin, D. Song, and B. Li, "Redcode: Risky code execution and generation benchmark for code agents," *Advances in Neural Information Processing Systems*, vol. 37, pp. 106 190–106 236, 2024.

[5] F. Qi, Y. Chen, M. Li, Y. Yao, Z. Liu, and M. Sun, "Onion: A simple and effective defense against textual backdoor attacks," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 9558–9566.

[6] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Y. Zhuo, D. Lo, and T. Chen, "Dece: Deceptive cross-entropy loss designed for defending backdoor attacks," *arXiv preprint arXiv:2407.08956*, 2024.

[7] C. Chen and J. Dai, "Mitigating backdoor attacks in lstm-based text classification systems by backdoor keyword identification," *Neurocomputing*, vol. 452, pp. 253–262, 2021.

[8] K. Chen, Y. Meng, X. Sun, S. Guo, T. Zhang, J. Li, and C. Fan, "Badpre: Task-agnostic backdoor attacks to pre-trained nlp foundation models," in *International Conference on Learning Representations*.

[9] K. Kurita, P. Michel, and G. Neubig, "Weight poisoning attacks on pretrained models," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2793–2806.

[10] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Y. Zhuo, and T. Chen, "Chain-of-thought in neural code generation: From and for lightweight language models," *IEEE Transactions on Software Engineering*, 2024.

[11] N. Jin, Z. Li, T. Zhang, and Q. Zeng, "Mscot: Structured chain-of-thought generation for multiple programming languages," *arXiv preprint arXiv:2504.10178*, 2025.

[12] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[13] W. Ahmad, M. G. R. Tushar, S. Chakraborty, and K.-W. Chang, "Avatar: A parallel corpus for java-python program translation," in *Findings of the Association for Computational Linguistics: ACL 2023*, 2023, pp. 2268–2281.

[14] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[15] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[16] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[17] N. Jain, A. Schwarzschild, Y. Wen, G. Somepalli, J. Kirchenbauer, P.-y. Chiang, M. Goldblum, A. Saha, J. Geiping, and T. Goldstein, "Baseline defenses for adversarial attacks against aligned language models," *arXiv preprint arXiv:2309.00614*, 2023.