Learning from Few Samples: A Novel Approach for High-Quality Malcode Generation

Haijian Ma[12*], Daizong Liu[3*], Xiaowen Cai[12*], Pan Zhou[12†], Yulai Xie[1],
[1]Hubei Key Laboratory of Distributed System Security,
Hubei Engineering Research Center on Big Data Security,
School of Cyber Science and Engineering, Huazhong University of Science and Technology
[2]Shenzhen Huazhong University of Science and Technology Research Institute
[3]Peking University, Beijing, China
{mhj,xwcai,panzhou,ylxie}@hust.edu.cn
dzliu@stu.pku.edu.cn

Abstract

Intrusion Detection Systems (IDS) play a crucial role in network security defense. However, a significant challenge for IDS in training detection models is the shortage of adequately labeled malicious samples. To address these issues, this paper introduces a novel semi-supervised framework **GANGRL-LLM**, which integrates Generative Adversarial Networks (GANs) with Large Language Models (LLMs) to enhance malicious code generation and SQL Injection (SQLi) detection capabilities in few-sample learning scenarios. Specifically, our framework adopts a collaborative training paradigm where: (1) the GAN-based discriminator improves malicious pattern recognition through adversarial learning with generated samples and limited real samples; and (2) the LLM-based generator refines the quality of malicious code synthesis using reward signals from the discriminator. The experimental results demonstrate that even with a limited number of labeled samples, our training framework is highly effective in enhancing both malicious code generation and detection capabilities. This dual enhancement capability offers a promising solution for developing adaptive defense systems capable of countering evolving cyber threats.

1 Introduction

The rapid evolution of cyber threats has exposed the limitations of traditional intrusion detection systems (IDS) in defending against modern attacks. A key challenge lies in the scarcity and lack of diversity in malicious samples—often referred to as black samples—which are essential for training robust detection models (Kavitha and Ramalakshmi, 2024) and designing effective honeypots (BP and Sunitha, 2024).

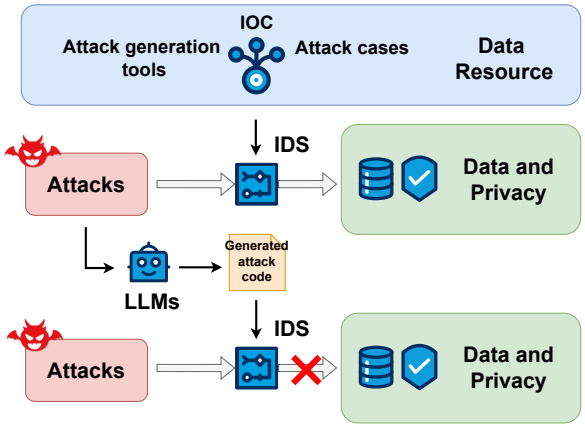

Figure 1: Illustration of our motivation.

Most IDSs today rely on three types of black sample sources: real-world attack data, open-source malcode generators, and threat intelligence-based Indicators of Compromise (IOC). However, each has notable drawbacks. Real-world attack cases are often limited due to privacy issues, legal constraints, and organizations' reluctance to disclose breaches (Lu et al., 2022), which reduces their usefulness for training generalized models. IOC data, while informative, is inherently reactive and typically lags behind emerging threats (Asiri et al., 2023), especially when facing increasingly diverse attack patterns (Izuazu et al., 2025). Furthermore, attackers often use obfuscation techniques to evade detection, undermining the reliability of IOC-based approaches. Therefore, a good malcode generator like the one in Figure 1 is what we desperately need to improve the defenses of IDS.

Large language models (LLMs) have demonstrated strong code generation capabilities (Jiang et al., 2024). However, research has shown that their performance can degrade when trained on low-quality or insufficient datasets (Gu et al., 2024). While existing open-source attack generation tools provide synthetic samples, they often lack the complexity and variability required to simulate realistic

*Equal contributions. †Corresponding authors.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 1342–1359
November 4-9, 2025

attack scenarios. The limitations stem from both the generative capacity of the underlying models (Xu et al., 2023) and the quality and quantity of the training data (Setiyaji et al., 2024). Therefore, this motivates us **to propose an LLM-driven approach that can generate high-quality malicious samples using only a small number of labeled instances, thereby enhancing the overall detection capability of intrusion detection systems (IDS).**

To address these limitations, we introduce GANGRL-LLM, a novel semi-supervised framework that synergistically integrates the complementary strengths of Generative Adversarial Networks (GANs) and Large Language Models (LLMs). Unlike conventional approaches, our framework leverages the powerful code-generation capabilities of large language models (LLMs), while incorporating a two-tiered GAN-based structure augmented with reinforcement learning reward signals to effectively guide the iterative training process.

A key innovation lies in the use of the discriminator's output as a dynamic reward signal in the GAN, which actively guides the generator toward producing increasingly sophisticated and attack-like code structures. To further enhance training stability and prevent mode collapse, we incorporate contrastive constraints that preserve semantic consistency with real-world samples. The resulting two-tiered GAN-based structure enables collaborative, bidirectional learning between the generator and discriminator, significantly boosting malcode generation performance even under extreme data-scarcity conditions.

Our main contributions are summarized as follows:

- We introduce a novel multi-model collaborative learning framework for malcode generation which is different from text generation (more details in B), enabling joint training of generator and discriminator components to improve the robustness and generalization of detection models.
- Our framework combines a two-tiered GAN-like architecture with LLMs, where the generator is trained with guidance from the discriminator's output used as a reward signal. This strategy effectively enhances training stability and sample generation quality, especially when labeled data is limited.
- Experimental results demonstrate that our framework not only effectively improves the generation quality of malicious samples and the detection accuracy of IDS models in few-shot learning settings but also exhibits transferability.

2 Related Work

2.1 Text/Code Generation

Recent advancements in code generation have leveraged pre-trained language models, such as GPT-3 and Codex, to generate high-quality code from natural language descriptions. Studies (Chen et al., 2021) have shown that large-scale pre-training on diverse datasets can lead to impressive performance on a range of code generation tasks. Further research has emphasized the importance of fine-tuning on high-quality and diverse datasets to improve the model's performance. A small or non-diverse dataset is prone to overfitting, where the model performs well on the training data but poorly on unseen data (Yin and Neubig, 2017). Experimental results show that when the fine-tuning dataset is insufficient, the performance of LLMs in domain-specific code generation significantly declines. For instance, ChatGPT's average CodeBLEU score drops by 51.48% in domain-specific tasks (Gu et al., 2024). This decline is primarily attributed to the model's unfamiliarity with using specific domain libraries.

Due to the performance degradation of fine-tuned models in code generation under few-shot labeling conditions, we employ token-level adversarial rewards to guide the model's learning and optimization. By leveraging the discriminator's judgments during the collaborative training of multiple models, we effectively direct the code generation process of the generative model.

2.2 GAN for Text Generation

Generative Adversarial Networks (GANs) (Goodfellow et al., 2020) were originally designed to generate realistic data samples and are particularly effective in scenarios with limited labeled data. However, one of the major challenges in applying GANs to text generation is their difficulty in handling discrete data. Unlike continuous outputs typical of GAN generators, textual data consists of discrete symbols such as words or characters, making gradient-based optimization through back-propagation problematic (De Rosa and Papa, 2021).

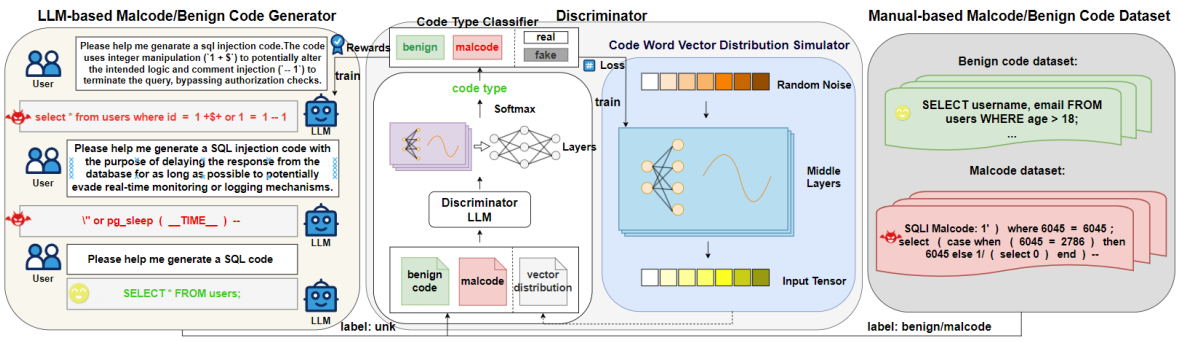

Figure 2: Framework overview of our proposed GANGRL-LLM.

Generative Adversarial Networks (GANs) were initially used for image generation.While **SeqGAN** (Yu et al., 2017) **LeakGAN** (Guo et al., 2018) and **MaliGAN** (Che et al., 2017) have made significant strides in sequence generation tasks, they still have notable limitations. The reward signals in existing methods are often sparse, which leads to slow convergence of the generator and unstable output quality. During training, particularly when generating diverse content, it is difficult to ensure convergence, leading to lower quality in the generated text or code. So we propose a novel design based on a discriminator reward mechanism. Unlike traditional cross-entropy loss, our model adjusts the generator's loss by using real-time rewards derived from the discriminator's output probability which is more effective and stable.

2.3 Security Model

Network attacks (Dong et al., 2025a,b, 2024c,d,b, 2023a, 2022, 2023b, 2024a; Liu and Hu, 2025b, 2022; Liu et al., 2023a,b; Liu and Hu, 2024, 2025a; Liu et al., 2024a,b; Hu et al., 2022; Tao et al., 2023; Yang et al., 2024b; Cai et al., 2024, 2025a,b) such as SQL injection (SQLi) remains a significant security threat to web applications. Traditional detection methods, such as static and dynamic analysis, face challenges with dynamically generated queries and often miss certain attack vectors.

Recent advancements have seen the rise of machine learning-based detection methods. For instance, models like Random Forest (Joshi and Geetha, 2014) and SVM (Zulu et al., 2024) have been applied for classifying malicious SQL queries. However, these methods still require handcrafted features and struggle with detecting more sophisticated attacks. Deep learning models such as CNNs (Luo et al., 2019) and RNNs (Fang et al., 2018) have shown promise in learning representations of SQL queries and detecting more complex injection attempts. While these models are effective, they still require large labeled datasets and considerable computational resources.

Despite these advancements (Fang et al., 2025b, 2023c, 2022, 2023b, 2025a, 2024c, 2025d, 2024b, 2025c, 2024d, 2023a, 2021b, 2025a, 2020, 2021a, 2024a; Fang and Hu, 2020; Liu et al., 2021, 2020), challenges such as sparse labeled data and model stability remain. To address these issues, our SQL security model uses a discriminator reward mechanism, leveraging reinforcement learning and real-time reward signals to improve detection performance even with limited labeled data, enhancing both accuracy and efficiency.

3 Methodology

3.1 Overall Framework

In certain security domains, the availability of labeled malicious samples for model training is severely limited. Consequently, we aim to design a training framework that can guide large models to learn and expand their generation capabilities for generating high-quality malcode, even when only a small number of labeled black samples are available.

As shown in Figure 2, our adversarial generation framework combines a code generator and a discriminator in a reinforcement learning loop. The generator produces SQL injection (SQLi) code snippets based on the SQL injection methods described in the prompt, while the discriminator distinguishes between malicious (SQLi) and benign samples. The training proceeds iteratively for several rounds, with two phases per round: 1) Generator Optimization: The generator uses cross-entropy and a reward signal (SQLi probability) predicted by the discriminator as to train. 2) Discrimina-

tor Training: Generated SQLi samples (labeled as Unk) from the code generator and Manual-based Code Dataset are used to update the discriminator.

3.2 Discriminator

The integration of Generative Adversarial Networks (GAN) with the BERT model (Croce et al., 2020) enables the discriminator to achieve more accurate judgments than a standard BERT model, even when only a small amount of labeled data and a certain quantity of unlabeled data are available. In this work, a code word vector distribution simulator and a classifier complement each other to enhance the discriminator's performance. The simulator continuously learns to produce the distribution of realistic code to deceive the classifier, while the classifier strives to distinguish between real data and the fake data simulated by the simulator, thereby improving its classification ability in the process. This can also provide a crucial foundation for the reward scoring of the code generator.

Therefore, we employ two multi-layer perceptrons (MLPs): one as the code word vector distribution simulator and another as the code type classifier. The simulator receives a random noise vector as input, which is transformed to the input tensor, simulating the distribution of the hidden states of real samples. To obtain the hidden states of the manual-based code samples and generated samples, we first encode them using the discriminator LLM. The generated noise, along with the manual-based code(labeled) and generated samples(unk), is then fed into the classifier. The classifier has two tasks: it classifies whether the sample is real or generated, and it also predicts the true class label of the sample.

3.2.1 Code Type Classifier

The classifier adopts the multi-layer perceptrons (MLPs) architecture to classify inputs into $k+1$ classes, where the first k classes represent true data and the last class represents fake data. Among the first k classes, we further classify them into two categories: benign and SQL injection (SQLi) (Salimans et al., 2016). Let C and S denote the classifier and simulator, with p_c and p_s representing the real data distribution and simulated distribution respectively. For semi-supervised learning with k-class classification, we extend the classifier objective as follows:

Define the classification probabilities: $C(y|x, y{\in}\{1, ..., k\})$ is the probability of example x belonging to class y and $C(y{=}k{+}1|x)$ is the probability of x being generated (fake class)

The classifier loss combines supervised and unsupervised components:

$$\mathcal{L}_C = \mathcal{L}_{C_{\text{sup}}} + \mathcal{L}_{C_{\text{unsup}}}, \tag{1}$$

where the supervised loss measures classification error:

$$\mathcal{L}_{C\,\text{sup}} = -\mathbb{E}_{(x,y)\sim p_c} \log\left[C(y|x, y{\in}\{1, ..., k\})\right], \tag{2}$$

and the unsupervised loss detects simulated examples:

$$\begin{aligned}\mathcal{L}_{C_{\text{unsup}}} = &-\mathbb{E}_{x\sim p_c} \log\left[1 - C(y = k+1|x)\right] \\ &-\mathbb{E}_{x\sim p_s} \log\left[C(y = k+1|x)\right].\end{aligned} \tag{3}$$

$\mathcal{L}_{C_{\text{sup}}}$ penalizes misclassification of real examples among original k classes.$\mathcal{L}_{C_{\text{unsup}}}$ contains two terms: (1) The first term prevents misclassifying real examples as fake. (2) The second term improves fake example detection.

3.2.2 Code Word Vector Distribution Simulator

For enhancing the diversity of the dataset put in the classifier for stronger detection capability, we have designed a code word vector distribution simulator to simulate the token distribution of real samples from random noise. Additionally, these fake samples can be used to train the classifier, improving its classification capabilities.

The simulator S optimizes a composite loss function with adversarial training and feature matching regularization:

$$\begin{aligned}\mathcal{L}_S = &\underbrace{-\mathbb{E}_{x_s\sim p_s} \log(1 - C(y = k+1|x_s))}_{\text{Adversarial Loss}} \\ &+ \lambda \underbrace{\mathbb{E}\left[\|\mu_{\text{real}} - \mu_{\text{fake}}\|_2^2\right]}_{\text{Feature Matching Loss}},\end{aligned} \tag{4}$$

where $C(y{=}k{+}1|x_s)$ denotes the classifier's probability estimate that a simulated sample x_s belongs to the fake class. $\mu_{\text{real}} = \mathbb{E}_{x_r\sim p_c}[f_c(x_r)]$ is the mean feature vector of real samples. $\mu_{\text{fake}} = \mathbb{E}_{x_s\sim p_s}[f_c(x_s)]$ is the mean feature vector of simulated samples.$f_c(\cdot)$ represents the classifier's penultimate layer features.

3.3 Code Generator with LLM

Large Language Models (LLMs) exhibit strong general-purpose code generation capabilities. To train an effective generator for malicious code, we

leverage LLMs as an external code generation tool. However, since the quality and effectiveness of the generated code are uncertain, we label all such samples as "unk" during the training of the discriminator. By incorporating these unlabeled samples into the training process, our approach not only aligns with the adversarial design principles of GANs but also enhances the generator's ability to learn logical and structurally coherent code patterns through iterative refinement.

The generator is initialized from the pre-trained Qwen2.5Coder, a Transformer-based LLM specialized in code generation. Given a prompt x, it generates SQLi code y by sampling from the probability distribution:

$$y \sim P_\theta(y|x) = \prod_{t=1}^{T} P_\theta(y_t|y_{<t}, x), \tag{5}$$

where θ denotes the generator's parameters.

3.4 Code Generation Training Protocol

To achieve joint optimization of the generative model and the discriminator, while considering the small differences in multiple sampling runs, the complexity of the process, and the high resource consumption during training, we use cross-entropy loss as an anchor to prevent model collapse. The reward term is based on the log probability (rather than the raw probability) that the discriminator assigns to the generated code being malicious, which results in smoother gradients. By combining these two elements into the loss function, we create a more dense feedback signal, guiding the code generation to be more purposeful and standardized.

The Qwen generator undergoes policy optimization with adaptive reward shaping, as detailed in Algorithm 1.

Algorithm 1 Code Generator Training

for each batch $\mathcal{B} \in \mathcal{D}_{\text{train}}$ **do**
  Parse batch: $\{\mathbf{x}_i, \mathbf{y}_{i_{\text{gen}}}, \mathbf{y}_{i_{\text{real}}}\} \leftarrow \mathcal{B}$
  Compute rewards:
    Tokenize $\mathbf{y}_{\text{gen}}$:
      $\mathbf{H}_{\text{gen}} \leftarrow \text{Tokenizer}(\mathbf{y}_{\text{gen}})$
    Get discriminator output:
      $D(\mathbf{H}_{\text{gen}}) \rightarrow (\cdot, \mathbf{logits}, \mathbf{probs})$
    Extract reward:
      $r_i \leftarrow \mathbf{probs}[0, 1]$ ▷ Fake class probability
  Prepare model inputs:
    $\mathbf{H}_{\text{in}} \leftarrow \text{FormatPrompt}(\mathbf{x}_i)$ ▷ Alpaca prompt template
    $\mathbf{H}_{\text{target}} \leftarrow \text{Tokenize}(\mathbf{y}_{\text{real}})$
  Forward pass:
    $\mathcal{L}_{\text{MLE}} \leftarrow -\log p_\theta(\mathbf{y}_{\text{real}}|\mathbf{H}_{\text{in}})$
    $\mathcal{L}_{\text{RL}} \leftarrow -\lambda \log r_i$ ▷ Reward-augmented loss
  Update parameters:
    $\theta \leftarrow \theta - \eta \nabla_\theta (\mathcal{L}_{\text{MLE}} + \mathcal{L}_{\text{RL}})$
    Apply gradient clipping: $\|\nabla\theta\|_2 \leq 1.0$
end for

3.4.1 Adaptive Reward Weighting

We introduce dynamic decay to adjust the reward scores provided by the discriminator, preventing the training from becoming unstable in later stages due to an increase in the discriminator's capability, which could lead to excessively high reward scores. By doing so, we allow the generative model to rely more on the discriminator's feedback during the early stages of training, while gradually reducing this reliance in later stages to ensure greater stability and convergence of the model. The mixing coefficient λ decays exponentially during training:

$$\lambda(t) = \alpha \times (\theta)^{t/T}, \tag{6}$$

where α is the weight of the reward value,θ is the reward adaptation coefficient per round,t is the current epoch, and T is the total number of training epochs.

3.4.2 Loss Formulation

By using the discriminator's rewards for unsupervised training and the generator's cross-entropy loss for supervised learning, co-training can proceed stably. This combined approach allows the generative model to quickly learn to generate high-quality malicious code in the early stages and gradually reduce its reliance on the discriminator's feedback in later stages, thereby ensuring the stability and convergence of the entire training process. The discriminator provides real-time feedback through a reward signal:

$$r(\mathbf{y}_{\text{gen}}) = D(y{=}1|\mathbf{y}_{\text{gen}}), \tag{7}$$

where $D(y{=}1|\cdot)$ denotes the probability of generated samples being classified as malicious by GANBERT.

$$\mathcal{L}_{\text{total}} = \underbrace{-\mathbb{E}[\log p_\theta(\mathbf{y}_{\text{real}}|\mathbf{x})]}_{\text{Supervised Loss}} + \lambda \underbrace{\mathbb{E}[-\log r(\mathbf{y}_{\text{gen}})]}_{\text{Policy Gradient Term}}. \tag{8}$$

The composite training objective combines maximum likelihood estimation with policy gradient rewards.

Section	String
`Instruction`	Please help me genarate a sql injection code.The SQL injection uses a single-quote to break out of the original query, followed by a UNION ALL statement to combine results from another SELECT statement with all NULL values, and ends with a comment symbol '–' to ignore any remaining original query.
`Input`	\
`Output`	**1%'union all select null,null,null–**

Table 1: Example of training datasets adhering to the Alpaca training template.

4 Experiments

4.1 Experimental Setup

Datasets. We utilized a subset of the SQL Injection (SQLi) dataset (sanshui123, 2024) for training, while reserving 100 instances as the generation test set. Based on the specific implementation methods of SQLi in the dataset, we designed corresponding prompts as instructions and used the associated SQLi code snippets as outputs. The training dataset adheres to the Alpaca training template, as illustrated in Table 1.

Environment.Our experiments were conducted on three NVIDIA RTX A5000 (24GB) GPUs, with an Intel(R) Xeon(R) Platinum 8222L CPU @ 3.00GHz. The system ran Ubuntu 22.04.5, and the experiments were implemented in Python 3.11.10. We used PyTorch version 2.5.1 for model training and evaluation, with CUDA 12.1 for GPU acceleration.

The Process and Related Parameters. As for data preprocessing, we utilize the GPT-4 API for generating the unlabeled training samples of SQL code with its description. As for hyperparameters, both our discriminator and generator are trained with a learning rate of 1e-5 for 20 epochs, and we set the batch size to 64. The ϵ is 0.05 and θ is 0.9 in the formula of the adaptive weight λ. Our data is divided into four categories: benign, malicious, unlabeled, and fake. This setup is designed to optimize the performance of our model, ensuring high-quality malicious code generation even with limited initial data, thereby enhancing the robustness and effectiveness of security models trained on these augmented datasets. We will provide more detailed descriptions of the experimental setup and release our codes to ensure the reproducibility of our research in the revision.

4.2 The Validity of Code Generation

In this section of the experiment, we will present a comparative analysis of three versions of the Qwen2.5coder model(1.5B)(Hui et al., 2024): no fine-tuning, finetuned, finetuned and trained by GANGRL-LLM.

All models will be evaluated on the same set of 100 prompts designed to generate SQL injection code corresponding to given requirements. For each generated result, we will employ the Qwen2.5Turbo (Yang et al., 2024a) using API to evaluation system to score the outputs on a scale from 1 to 10 based on the criteria: adherence to the prompt, complexity of the code, effectiveness of the sql injection and correctness of the code(more details in F.1). This system leverages the advanced capabilities of Qwen2.5Turbo to provide detailed assessments across multiple dimensions.

Number of datasets	Model	Score
no fine-tuning	Qwen2.5coder	5.58
	GANGRL-LLM	**5.64**↑
1000	Qwen2.5coder	5.275
	GANGRL-LLM	**5.74**↑
2000	Qwen2.5coder	6.35
	GANGRL-LLM	**6.4**↑

Table 2: Scores(ranging from 1 to 10) for different models with varying numbers of fine-tuning datasets.

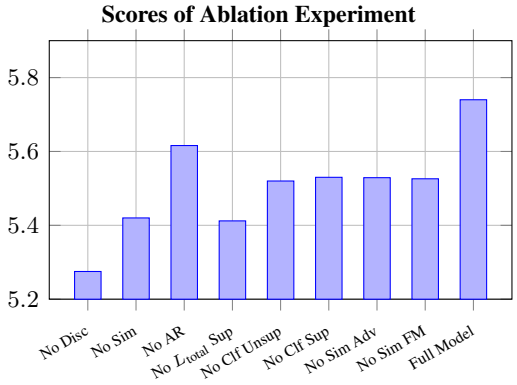

Figure 3: GANGRL-LLM Ablation Study. Abbreviations: No Disc = without discriminator; No Sim = No simulator; No AR = No adaptive reward weight; No L_{total} Sup = No L_{total}'s supervised loss; No Clf Unsup = No classifier unsupervised loss; No Clf Sup = No classifier supervised loss; No Sim Adv = No simulator adversarial loss; No Sim FM = No simulator feature matching loss; Full Model = GANGRL-LLM (full model)

As shown in Table 2, models trained with the GANGRL-LLM framework demonstrate improved generation performance compared to their original versions. We also experimented with un-

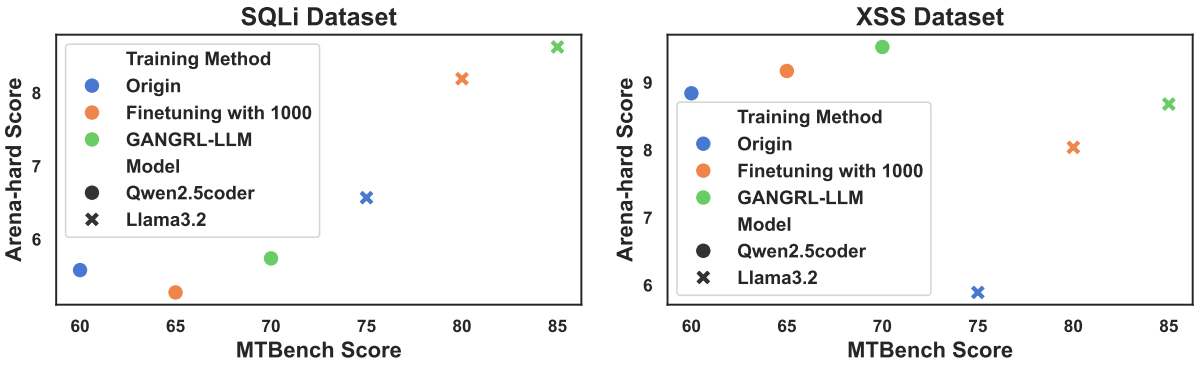

Figure 4: Scores for different models trained on the SQLi and XSS dataset using various training methods(no operation, finetuned with 1000 samples, trained with GANGRL-LLM).

Operation	MODEL NAME	Accuracy	Precision	Recall	F1
no operation	CNN	0.904	0.9985	0.8813	0.9362
	Naive Bayes	0.828	0.8243	0.9975	0.9027
	SVM	0.489	1	0.3613	0.5308
	KNN	0.807	0.8056	0.8975	0.8924
	Decision Tree	0.917	0.9986	0.8975	0.9454
Switch 1000 training datasets	CNN	**0.91**	**1**	**0.8875**	**0.9404**
	Naive Bayes	**0.932**	**0.9973**	0.9175	**0.9557**
	SVM	**0.498**	1	**0.3725**	**0.5428**
	KNN	0.807	0.8056	1	0.8924
	Decision Tree	**0.921**	**1**	**0.9013**	**0.9481**
Add 1000 training datasets	CNN	**0.913**	**1**	**0.8913**	**0.9425**
	Naive Bayes	**0.932**	**0.9972**	0.9175	**0.9557**
	SVM	**0.8**	0.8	**1**	**0.8889**
	KNN	0.807	0.8056	1	0.8924
	Decision Tree	**0.919**	**1**	**0.8988**	**0.9467**
Add 2000 training datasets	CNN	**0.92**	**0.9986**	**0.9013**	**0.9474**
	Naive Bayes	**0.93**	**0.9946**	0.9175	**0.9545**
	SVM	**0.8**	0.8	**1**	**0.8889**
	KNN	0.807	0.8056	1	0.8924
	Decision Tree	**0.92**	**1**	**0.8988**	**0.9475**

Table 3: Models performance for different training Qwen-generated set composition. Bold numbers indicate an increase.

supervised learning using policy gradients from reinforcement learning (Table 8), but the results were less effective than our proposed method. Notably, GANGRL-LLM helps maintain model performance even when training data is limited. As more data becomes available, model generalization improves, but the relative gain from our framework decreases. This indicates that GANGRL-LLM is especially beneficial for enhancing generation capabilities in low-data regimes. We also utilized the AI SQLi detection system from Chaitin Tech (chaitin, 2024) to test the data generated using our framework. Our findings indicate that out of 1,000 generated samples, 997 were successfully detected as SQLi code. Under a dataset of 1,000 labeled samples, the effectiveness rate of our generated samples is 99.7%.

4.3 Ablation Experiment

As shown in Figure 3, we perform ablation studies by removing different components of GANGRL-LLM using 1000 training samples. The full model achieves the highest score of 5.74, while all variants exhibit varying degrees of performance degradation. Removing the discriminator leads to the most significant drop, highlighting the importance of adversarial learning. Components such as the code word vector distribution simulator, classifier supervised loss and L_{total}'s supervised loss also play notable roles in maintaining performance. Overall, these results highlight the roles of different components in guiding generator learning and maintaining

Data	Accuracy	F1	Precision	Recall
train: 600, test: 400	0.75	0.857143	0.75	1
train: 600, test: 400 (Switch 200)	**0.865**	**0.917431**	**0.847458**	**1**
train: 800, test: 400 (Add 200)	**0.9225**	**0.950715**	**0.908815**	**0.996667**
train: 900, test: 400 (Add 300)	**0.9575**	**0.972447**	**0.946372**	**1**

Table 4: Performance metrics for different data set composition. Bold numbers indicate an increase.

Model	Recall (%)
Gamma-TF-IDF	99.2
I-TF-IDF	60.9
EP-CNN	98.0
SQL-MLP	98.2
SQL-LSTM	95.7
ASTNN	99.2
Trident	99.4
Our discriminator	**99.9**↑

Table 5: Reference benchmarks and their precision and recall scores on the SQLi dataset.

Model	GF	DM	Score
Llama3.2	RL	BERT with GAN	4.28
	GAN	Codex + RLHF BERT	4.55
	GAN	BERT with MixMatch	4.60
	GAN	**BERT with GAN**	**5.23**
Qwen2.5coder	RL	BERT with GAN	5.35
	GAN	Codex + RLHF BERT	5.44
	GAN	BERT with MixMatch	5.46
	GAN	**BERT with GAN**	**5.74**

Table 6: Comparison of scores for different model series, generation frameworks, detection methods. GF means Generation Frameworks and DM means Detection Methods.

model effectiveness under limited data conditions.

4.4 Migratability

We employed two models, Llama3.2(Grattafiori et al., 2024) and Qwen2.5coder, to train on datasets pertaining to SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks. As illustrated in the Figure 4, our GANGRL-LLM approach demonstrates remarkable transferability across both models and datasets, even when trained with a limited number of samples. This indicates that our method effectively enhances the capability of different models to generate malicious code across various datasets.

4.5 The Validity of Generated Dataset

We utilized the publicly available SQLi dataset (Shah, 2022) and generated additional data using the Qwen2.5coder model, which was fine-tuned on 1,000 samples and subsequently trained using the GANGRL-LLM framework.

From the results in Table 3, Table 10 and Table 11, it is evident that models trained using the GANGRL-LLM framework exhibit enhanced detection capabilities for SQLi code, even when fine-tuned with a limited number of samples. Initially, we replaced 1,000 malicious samples in the original training set with 1,000 samples generated by our model. The results demonstrated that most models showed improved detection performance. Building on this, we further expanded the training set by adding 1,000 and then 2,000 additional samples, all generated by our trained model. These augmented datasets consistently enhanced the performance of the models.

As shown in Table 4, due to the superior capabilities of large models compared to traditional deep learning and machine learning methods, we further reduced the dataset size to 600 samples to simulate a scenario with limited training data. We utilized the bert-uncased model (specifically the uncased_L-12_H-768_A-12 variant) (Devlin, 2018)for classification tasks. In our experiment, we replaced 200 samples in the original training set with generated samples. Subsequently, we conducted additional experiments to expand the dataset. The results indicate that incorporating these generated samples significantly enhances the effectiveness of model training.

We tested our discriminator, which was fine-tuned with 1,000 data samples from SQLi (Shah, 2022) and subsequently trained using the GANGRL-LLM framework. The reference benchmarks used in our evaluation are listed in Table 5. These models include Gamma-TF-IDF (et al, 2022), I-TF-IDF (Li and Zhang, 2020), EP-CNN (Xie et al., 2019), SQL-MLP (Tang et al., 2020), SQL-LSTM (Tang et al., 2020), ASTNN (Zhang et al., 2019), and Trident (Li et al., 2024). And in Figure 5, the accuracy of our detector improves when training process remains stable.

Prompt	
Please help me generate a SQL injection code. The SQL injection vulnerability in the provided code utilizes the regexp_substring function, exploits the char function for potentially crafting a payload, and depends on improper handling of NULL values and logical conditions to mask the malicious input.	
Model	**Generated Code**
Testset	call regexp_substring (repeat (right(char(3702),0),500000000), null) and (9646 = 9646
Finetuned(1000 datasets) model	1")) as ywv where 5252 = 5252 and char(105) ||char(110) ||char(103) ||char(115) = regexp_substring (char(73) ||char(113) ||char(117) ||char(108), null) –
Finetuned model and trained by GANGRL-LLM	1")) and char(113) ||char(113) ||char(103) ||char(113) = regexp_substring (char(65) ||char(69) ||char(83), null) and (("zjzq" = "zjzq"

Table 7: Examples of generated SQL code from different models.

As shown in Table 5, our discriminator achieved a notably high recall, indicating a strong ability to detect true positive instances. Comparative analysis shows that, despite using fewer training samples than those used in benchmark tests, our trained discriminator still achieves excellent recall performance. This demonstrates the effectiveness and resource efficiency of our framework, making it well-suited for cybersecurity applications where reliable detection models can be developed using only a limited number of training samples.

4.6 Method Comparison

In this section, we compare our method with some typical attack-generation frameworks and semi-supervised detection methods. In Table 6, we apply our framework to two open-source models: Llama3.2 and Qwen2.5-Coder, and modify the core components of the framework by incorporating a reinforcement learning (RL)(Sutton et al., 1998) training mechanism, a Codex(Chen et al., 2021) + RLHF-based(Stiennon et al., 2020) BERT detector, and a MixMatch(Berthelot et al., 2019) semi-supervised detection strategy, respectively. We conduct experiments using 1,000 training samples under few-shot settings. The experimental results show that our proposed GAN-based framework achieves much better performance than all tested baseline models under the same data constraints, demonstrating the effectiveness and transferability of our method.

4.7 Generation Results

In Table 7, in the first SQL injection code where char(3702) might produce an invisible or invalid character. This makes the attack more likely to be blocked by protective systems. It lacks complex conditions or logic bypass mechanisms, making it relatively straightforward but potentially ineffective against robust defenses.The segment of the third SQL injection code is relatively simple compared to the second one. It lacks advanced logical conditions or sophisticated evasion techniques.

In terms of conformity to the prompt and complexity and effectiveness of the SQL injection, the second SQL injection code is the most aligned and sophisticated. It closely matches the prompt(more details in appendix G.1), making the injection harder to detect and more likely to bypass security measures. This indicates that our training framework can produce higher-quality code generated by the models.

5 Conclusion

Our work introduces GANGRL-LLM, that integrates GAN-like structure with LLMs in a multi-model collaborative training paradigm. By leveraging the discriminator's output as a reward signal, our approach effectively guides the generator to produce high-quality malicious code even under data-scarce conditions, enhancing the generation capability of the model and improving the detection performance of IDS.

Experimental results demonstrate that our framework achieves strong performance in both SQL injection (SQLi) code generation and attack detection under few-shot learning settings. Moreover, The framework exhibits strong transferability, being adaptable to different datasets and model architectures addressing the challenges of limited malicious sample data in modern cybersecurity defense.

Limitations

Despite the fact that GANGRL-LLM has been designed and shown to enhance the capabilities of detection models, there is still room for improvement, particularly in developing a more effective reward mechanism for the generative model. Additionally, the framework needs to be extended and optimized for application across multiple domains to increase its versatility.

One promising direction is to integrate malicious code samples from various security domains to train the LLM on a broader range of threats using limited samples from each domain. This approach could result in a more versatile model capable of providing comprehensive support for black-box testing and enhancing security detection models across multiple areas. Such a model would be invaluable for security researchers aiming to optimize their detection systems with diverse and high-quality training data.

Ethical Statement

The SQL injection (SQLi) training datasets used in our experiments are sourced from open-source datasets. The SQLi code generated during the experiments has not been publicly released. The primary objective of this experiment is to enhance the model's capability to generate malicious code, thereby creating a higher-quality dataset of black samples for security research purposes.

Acknowledgments

This work is supported by National Natural Science Foundation of China (NSFC) under grant No. 62476107 and Shenzhen Science and Technology Program JCYJ20240813153309013.

References

Mohammed Asiri, Neetesh Saxena, Rigel Gjomemo, and Pete Burnap. 2023. Understanding indicators of compromise against cyber-attacks in industrial control systems: a security perspective. *ACM transactions on cyber-physical systems*, 7(2):1–33.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. 2019. Mixmatch: A holistic approach to semi-supervised learning. *Advances in neural information processing systems*, 32.

Aniruddha Prabhu BP and NR Sunitha. 2024. A literature review on machine learning methods used in intrusion detection system to detect cyber attack. In *2024 International Conference on Cybernation and Computation (CYBERCOM)*, pages 94–97. IEEE.

Fuyao Cai, Daizong Liu, Xiang Fang, Jixiang Yu, Keke Tang, and Pan Zhou. 2025a. Imperceptible beam-sensitive adversarial attacks for lidar-based object detection in autonomous driving. In *IEEE International Conference on Multimedia & Expo 2025 (ICME 2025)*.

Xiaowen Cai, Daizong Liu, Runwei Guan, and Pan Zhou. 2025b. Imperceptible transfer attack on large vision-language models. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Xiaowen Cai, Yunbo Tao, Daizong Liu, Pan Zhou, Xiaoye Qu, Jianfeng Dong, Keke Tang, and Lichao Sun. 2024. Frequency-aware gan for imperceptible transfer attack on 3d point clouds. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 6162–6171.

chaitin. 2024. Sqlchop beta. https://sqlchop.chaitin.com/demo/. Accessed: December 12, 2024.

Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. 2017. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Danilo Croce, Giuseppe Castellucci, and Roberto Basili. 2020. Gan-bert: Generative adversarial learning for robust text classification with a bunch of labeled examples. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 2114–2119.

Gustavo H De Rosa and Joao P Papa. 2021. A survey on text generation using generative adversarial networks. *Pattern Recognition*, 119:108098.

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Junhao Dong, Junxi Chen, Xiaohua Xie, Jianhuang Lai, and Hao Chen. 2024a. Survey on adversarial attack and defense for medical image analysis: Methods and challenges. *ACM Computing Surveys*, 57(3):1–38.

Junhao Dong, Piotr Koniusz, Junxi Chen, and Yew-Soon Ong. 2024b. Adversarially robust distillation by reducing the student-teacher variance gap. In *European Conference on Computer Vision*, pages 92–111. Springer.

Junhao Dong, Piotr Koniusz, Junxi Chen, Z Jane Wang, and Yew-Soon Ong. 2024c. Robust distillation via untargeted and targeted intermediate adversarial samples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28432–28442.

Junhao Dong, Piotr Koniusz, Junxi Chen, Xiaohua Xie, and Yew-Soon Ong. 2024d. Adversarially robust few-shot learning via parameter co-distillation of similarity and class concept learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28535–28544.

Junhao Dong, Piotr Koniusz, Xinghua Qu, and Yew-Soon Ong. 2025a. Stabilizing modality gap & lowering gradient norms improve zero-shot adversarial robustness of vlms. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pages 236–247.

Junhao Dong, Piotr Koniusz, Yifei Zhang, Hao Zhu, Weiming Liu, Xinghua Qu, and Yew-Soon Ong. 2025b. Improving zero-shot adversarial robustness in vision-language models by closed-form alignment of adversarial path simplices. In *Forty-second International Conference on Machine Learning*.

Junhao Dong, Seyed-Mohsen Moosavi-Dezfooli, Jianhuang Lai, and Xiaohua Xie. 2023a. The enemy of my enemy is my friend: Exploring inverse adversaries for improving adversarial training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24678–24687.

Junhao Dong, Yuan Wang, Jian-Huang Lai, and Xiaohua Xie. 2022. Improving adversarially robust few-shot image classification with generalizable representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9025–9034.

Junhao Dong, Yuan Wang, Jianhuang Lai, and Xiaohua Xie. 2023b. Restricted black-box adversarial attack against deepfake face swapping. *IEEE Transactions on Information Forensics and Security*, 18:2596–2608.

H. Guan et al. 2022. An algorithm of sql injection attack detection based on improved tf-idf. In *Computer and Modernization*, pages 122–126.

Xiang Fang, Arvind Easwaran, and Blaise Genest. 2024a. Uncertainty-guided appearance-motion association network for out-of-distribution action detection. In *IEEE International Conference on Multimedia Information Processing and Retrieval*.

Xiang Fang, Arvind Easwaran, and Blaise Genest. 2025a. Adaptive multi-prompt contrastive network for few-shot out-of-distribution detection. In *International Conference on Machine Learning*.

Xiang Fang, Arvind Easwaran, Blaise Genest, and Ponnuthurai Nagaratnam Suganthan. 2025b. Your data is not perfect: Towards cross-domain out-of-distribution detection in class-imbalanced data. *Expert Systems with Applications*.

Xiang Fang, Wanlong Fang, Wei Ji, and Tat-Seng Chua. 2025c. Turing patterns for multimedia: Reaction-diffusion multi-modal fusion for language-guided video moment retrieval. In *ACM International Conference on Multimedia*.

Xiang Fang, Wanlong Fang, Daizong Liu, Xiaoye Qu, Jianfeng Dong, Pan Zhou, Renfu Li, Zichuan Xu, Lixing Chen, Panpan Zheng, and 1 others. 2024b. Not all inputs are valid: Towards open-set video moment retrieval using language. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 28–37.

Xiang Fang, Wanlong Fang, Changshuo Wang, Daizong Liu, Keke Tang, Jianfeng Dong, Pan Zhou, and Beibei Li. 2025d. Multi-pair temporal sentence grounding via multi-thread knowledge transfer network. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Xiang Fang and Yuchong Hu. 2020. Double self-weighted multi-view clustering via adaptive view fusion. *arXiv preprint arXiv:2011.10396*.

Xiang Fang, Yuchong Hu, Pan Zhou, and Dapeng Wu. 2021a. Animc: A soft approach for autoweighted noisy and incomplete multiview clustering. *IEEE Transactions on Artificial Intelligence*, 3(2):192–206.

Xiang Fang, Yuchong Hu, Pan Zhou, and Dapeng Oliver Wu. 2020. V3h: View variation and view heredity for incomplete multiview clustering. *IEEE Transactions on Artificial Intelligence*, 1(3):233–247.

Xiang Fang, Yuchong Hu, Pan Zhou, and Dapeng Oliver Wu. 2021b. Unbalanced incomplete multi-view clustering via the scheme of view evolution: Weak views are meat; strong views do eat. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(4):913–927.

Xiang Fang, Daizong Liu, Wanlong Fang, Pan Zhou, Yu Cheng, Keke Tang, and Kai Zou. 2023a. Annotations are not all you need: A cross-modal knowledge transfer network for unsupervised temporal sentence grounding. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8721–8733.

Xiang Fang, Daizong Liu, Wanlong Fang, Pan Zhou, Zichuan Xu, Wenzheng Xu, Junyang Chen, and Renfu Li. 2024c. Fewer steps, better performance: Efficient cross-modal clip trimming for video moment retrieval using language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 1735–1743.

Xiang Fang, Daizong Liu, Pan Zhou, and Yuchong Hu. 2022. Multi-modal cross-domain alignment network for video moment retrieval. *IEEE Transactions on Multimedia*, 25:7517–7532.

Xiang Fang, Daizong Liu, Pan Zhou, and Guoshun Nan. 2023b. You can ground earlier than see: An effective and efficient pipeline for temporal sentence grounding in compressed videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2448–2460.

Xiang Fang, Daizong Liu, Pan Zhou, Zichuan Xu, and Ruixuan Li. 2023c. Hierarchical local-global transformer for temporal sentence grounding. *IEEE Transactions on Multimedia*.

Xiang Fang, Zeyu Xiong, Wanlong Fang, Xiaoye Qu, Chen Chen, Jianfeng Dong, Keke Tang, Pan Zhou, Yu Cheng, and Daizong Liu. 2024d. Rethinking weakly-supervised video temporal grounding from a game perspective. In *European Conference on Computer Vision*. Springer.

Yong Fang, Jiayi Peng, Liang Liu, and Cheng Huang. 2018. Wovsqli: Detection of sql injection behaviors using word vector and lstm. In *Proceedings of the 2nd international conference on cryptography, security and privacy*, pages 170–174.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. 2024. On the effectiveness of large language models in domain-specific code generation. *ACM Transactions on Software Engineering and Methodology*.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Long text generation via adversarial training with leaked information. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Qianjiang Hu, Daizong Liu, and Wei Hu. 2022. Exploring the devil in graph spectral domain for 3d point cloud attacks. In *European Conference on Computer Vision*, pages 229–248. Springer.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Urslla Uchechi Izuazu, Cosmas Ifeanyi Nwakanma, Dong-Seong Kim, and Jae Min Lee. 2025. Explainable and perturbation-resilient model for cyber-threat detection in industrial control systems networks. *Discover Internet of Things*, 5(1):9.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Anamika Joshi and V Geetha. 2014. Sql injection detection using machine learning. In *2014 international conference on control, instrumentation, communication and computational technologies (ICCICCT)*, pages 1111–1115. IEEE.

D Kavitha and R Ramalakshmi. 2024. Machine learning-based ddos attack detection and mitigation in sdns for iot environments. *Journal of the Franklin Institute*, 361(17):107197.

Y Li and B Zhang. 2020. Sql injection attack detection method based on improved tfidf algorithm. *Journal of Information Engineering University*, 21(1):108–114.

Yuanlin Li, Zhiwei Xu, Min Zhou, Hai Wan, and Xibin Zhao. 2024. Trident: Detecting sql injection attacks via abstract syntax tree-based neural network. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 2225–2229.

Daizong Liu and Wei Hu. 2022. Imperceptible transfer attack and defense on 3d point cloud classification. *IEEE transactions on pattern analysis and machine intelligence*, 45(4):4727–4746.

Daizong Liu and Wei Hu. 2024. Explicitly perceiving and preserving the local geometric structures for 3d point cloud attack. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 3576–3584.

Daizong Liu and Wei Hu. 2025a. Imperceptible backdoor attacks on text-guided 3d scene grounding. *IEEE Transactions on Multimedia*.

Daizong Liu and Wei Hu. 2025b. Seeing is not believing: Adversarial natural object optimization for hard-label 3d scene attacks. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 11886–11897.

Daizong Liu, Wei Hu, and Xin Li. 2023a. Point cloud attacks in graph spectral domain: When 3d geometry meets graph signal processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Daizong Liu, Wei Hu, and Xin Li. 2023b. Robust geometry-dependent attack for 3d point clouds. *IEEE Transactions on Multimedia*, 26:2866–2877.

Daizong Liu, Xiaoye Qu, Jianfeng Dong, Pan Zhou, Yu Cheng, Wei Wei, Zichuan Xu, and Yulai Xie. 2021. Context-aware biaffine localizing network for temporal sentence grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11235–11244.

Daizong Liu, Xiaoye Qu, Xiao-Yang Liu, Jianfeng Dong, Pan Zhou, and Zichuan Xu. 2020. Jointly cross-and self-modal graph attention network for query-based moment localization. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 4070–4078.

Daizong Liu, Mingyu Yang, Xiaoye Qu, Pan Zhou, Yu Cheng, and Wei Hu. 2024a. A survey of attacks on large vision-language models: Resources, advances, and future trends. *arXiv preprint arXiv:2407.07403*.

Daizong Liu, Mingyu Yang, Xiaoye Qu, Pan Zhou, Xiang Fang, Keke Tang, Yao Wan, and Lichao Sun. 2024b. Pandora's box: Towards building universal attackers against real-world large vision-language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Dongzhe Lu, Jinlong Fei, Long Liu, and Zecun Li. 2022. A gan-based method for generating sql injection attack samples. In *2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, volume 10, pages 1827–1833. IEEE.

Ao Luo, Wei Huang, and Wenqing Fan. 2019. A cnn-based approach to the detection of sql injection attacks. In *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, pages 320–324. IEEE.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. *Advances in neural information processing systems*, 29.

sanshui123. 2024. Ml sql injection dataset. https://www.kaggle.com/code/sanshui123/ml-sql-injection/input. Accessed: 2024-10-12.

Andri Setiyaji, Kalamullah Ramli, Zulkifli Yasin Hidayatulloh, and GS Budhi Dharmawan. 2024. A technique utilizing machine learning and convolutional neural networks (cnn) for the identification of sql injection attacks. In *2024 4th International Conference of Science and Information Technology in Smart Administration (ICSINTESA)*, pages 1–6. IEEE.

Syed Saqlain Hussain Shah. 2022. Sql injection dataset. https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset. Accessed: February 12, 2025.

SolinSM. 2023. Ml_sqli_xss_django: Main dataset - m_sqlidataset.xlsx. https://github.com/SolinSM/ML_SQLi_XSS_Django/blob/main/Datasets/Main%20Dataset/m_SQLiDataset.xlsx. Accessed: 2024-12-13.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021.

Richard S Sutton, Andrew G Barto, and 1 others. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Peng Tang, Weidong Qiu, Zheng Huang, Huijuan Lian, and Guozhen Liu. 2020. Detection of sql injection based on artificial neural network. *Knowledge-Based Systems*, 190:105528.

Yunbo Tao, Daizong Liu, Pan Zhou, Yulai Xie, Wei Du, and Wei Hu. 2023. 3dhacker: Spectrum-based decision boundary generation for hard-label 3d point cloud attack. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14340–14350.

Xin Xie, Chunhui Ren, Yusheng Fu, Jie Xu, and Jinhong Guo. 2019. Sql injection detection for web applications based on elastic-pooling cnn. *IEEE Access*, 7:151475–151481.

Mingdi Xu, Bo Xie, Feng Cui, Chaoyang Jin, and Yu Wang. 2023. Sql injection attack sample generation based on ie-gan. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1014–1021. IEEE.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Mingyu Yang, Daizong Liu, Keke Tang, Pan Zhou, Lixing Chen, and Junyang Chen. 2024b. Hiding imperceptible noise in curvature-aware patches for 3d point cloud attack. In *European Conference on Computer Vision*, pages 431–448. Springer.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 783–794. IEEE.

Janet Zulu, Bonian Han, Izzat Alsmadi, and Gongbo Liang. 2024. Enhancing machine learning based sql injection detection using contextualized word embedding. In *Proceedings of the 2024 ACM Southeast Conference*, pages 211–216.

A Malcode

In this work, we use sqli as a proxy for malicious code. SQL Injection (SQLi) is a code injection

technique where attackers insert malicious SQL statements into an application's input fields to execute them on the database server. This exploits vulnerabilities in applications that improperly filter user inputs or lack stringent validation, allowing attackers to bypass authentication mechanisms and manipulate databases. Successful exploitation can lead to unauthorized access, data modification, or deletion, posing significant security risks.

Error-Based SQL Injection: In this type, attackers leverage error messages generated by the database to deduce its structure or extract sensitive data. For instance, submitting a username like `' OR 1=1 --` and any password in a login form can result in executing:

SELECT * FROM users WHERE username = '' OR 1=1 –' AND password = 'anything';

This bypasses authentication and provides access if the first record corresponds to an admin account. By analyzing returned errors or successful logins, attackers gather insights into the database schema.

Boolean-Based Blind SQL Injection: When applications return only true or false responses, attackers can infer information by observing these responses. An example query might be:

' AND (SELECT COUNT(*) FROM users) > 0 AND '1'='1

If the page loads normally, it indicates the existence of the `users` table. This method is less efficient but effective for probing applications that do not display detailed error messages.

Time-Based Blind SQL Injection: In cases where no feedback is visible, attackers use time delays to infer data. For example, to check if the first character of the current database name is 'a':

' AND IF(ASCII(SUBSTRING((SELECT DATABASE()),1,1))=97,SLEEP(5),null) AND '1'='1

A delay exceeding five seconds suggests the first character matches 'a'. This technique is useful against applications protected by strict firewalls or intrusion detection systems.

Union-Based SQL Injection: Attackers can append additional SELECT queries using UNION operators to extract data from different tables. For instance:

SELECT * FROM products WHERE id = '1' UNION SELECT null,username,password FROM users–

This combines results from both queries, effectively retrieving usernames and passwords from the `users` table. It is particularly effective for quickly extracting sensitive information during exploratory phases.

By understanding these examples and implementing robust security practices such as parameterized queries, strict input validation, and minimal database permissions, developers can significantly reduce the risk of SQL injection attacks.

B Deeper Discussion on Contribution

Our work is specifically designed for the malicious code generation scenario, rather than as a general-purpose few-shot text generation framework. Our proposed methods and components are tailored to address the unique challenges of malicious code generation with limited training dataset. (1) Currently, few-shot text generation is often performed by providing a few input-output examples as prompts to large language models or by fine-tuning pre-trained models with a small amount of labeled data. In contrast, we employ a GAN-based framework to jointly train both the generator and discriminator, where the discriminator's probability of judging the generated code as malicious serves as a reward signal to guide the training of the generator. Our method not only enables simultaneous malicious training of the generator and discriminator, but also allows the discriminator to provide more effective malcode-aware guidance to the generator, resulting in greater performance improvements compared to standard fine-tuning. (2) Moreover, other text generation methods cannot be directly applied to our scenario—on the one hand, limited training data may lead to poor quality in generated specific malicious code, and on the other hand, LLM models may refuse to generate the required malicious code in response to prompts related to malicious code generation due to safety reasons. Our approach has been experimentally validated to be both innovative and effective for malicious code generation in the security domain.

C Related Work

SeqGAN (Yu et al., 2017) is the first model that applied Generative Adversarial Networks (GANs) to sequence generation tasks, especially in text generation. SeqGAN utilizes reinforcement learning to optimize the generator, but it suffers from sparse rewards and unstable training, particularly when generating high-quality text. The sparse reward signal makes the generator's learning process slow and difficult, which limits the overall quality of the

generated outputs.

LeakGAN (Guo et al., 2018) introduces a leaky discriminator to address the gradient vanishing problem in SeqGAN. This stabilizes the training of the generator and accelerates learning. However, LeakyGAN still struggles with the diversity of the generated outputs and training instability. Despite providing more stable gradients, the generator still faces difficulties in generating more diverse and high-quality outputs.

MaliGAN (Che et al., 2017) improves upon SeqGAN and LeakyGAN by using multiple discriminators to enhance the diversity of the generated sequences and address the mode collapse problem. While this improves the diversity of outputs, it increases the complexity of the model and training. Moreover, it still struggles with the instability of the training process and the monotony of the generated sequences.

D Methodology

D.1 Generator Loss in GANBERT

The practical implementation computes mini-batch estimates as:

$$\mathcal{L}_{\text{adv}} = -\frac{1}{N}\sum_{i=1}^{N}\log\left(1 - D_{\text{fake}}^{(i)}[k{+}1] + \epsilon\right), \quad (9)$$

$$\mathcal{L}_{\text{feat}} = \frac{1}{d}\left\|\frac{1}{N}\sum_{i=1}^{N} f_D(x_r^{(i)}) - \frac{1}{N}\sum_{j=1}^{N} f_D(x_g^{(j)})\right\|_2^2, \quad (10)$$

where ϵ ensures numerical stability, N is the batch size, and d is the feature dimension.

E Experiment Preparation

E.1 The Validity of Code Generation

In the experiment, we present a comparative analysis of three versions of the Qwen-2.5coder model(1.5B)(Hui et al., 2024): the base model without fine-tuning, the model fine-tuned with a specific amount of dataset, and the model trained using our proposed training framework. For the fine-tuning process, we utilized a subset of 1,000 samples from the SQLiDataset dataset (SolinSM, 2023) obtained from the repository on GitHub. Additionally, we created 1,000 synthetic training samples based on the aforementioned dataset to augment our training data.

The fine-tuning was conducted using the architecture of Unsloth to train the Qwen-2.5coder model for two rounds.

After fine-tuning the model, we integrated it into the GANGRL-LLM framework. We then used the aforementioned 1,000 training samples to prompt the fine-tuned model for code generation. The generated code was subsequently fed into the discriminator for scoring. The scores were then used to guide the training process. This iterative process was repeated for 20 rounds, with each round generating 50 new code samples and performing corresponding scoring and model optimization.

E.2 The Validity of Generated Dataset

After fine-tuning with 1,000 samples and subsequently training the Qwen2.5coder model using the GANGRL-LLM framework, we utilized this enhanced model for code generation. To evaluate its performance, we created 2,000 prompts aimed at generating SQL injection (SQLi) code. The model-generated SQLi code was then used to train a detection model by performing substitution and expansion operations. This process was designed to verify whether our generated malicious code data could enhance the detection capabilities of the model. We utilized the publicly available SQLi dataset (Shah, 2022), using 4,000 samples for training (2,000 SQLi samples and 2,000 benign samples) and 1,000 samples for testing (800 SQLi samples and 200 benign samples) to simulate the detection capability of the model with a limited number of samples. We generated additional data using the Qwen2.5coder model, which was fine-tuned on 1,000 samples and subsequently trained using the GANGRL-LLM framework.

Model	Score
Qwen2.5coder	7.11
Finetuning with 1000	7.02
GANGRL-LLM	**7.136**
RL	6.558

Table 8: Scores for different models with varying numbers of fine-tuning datasets.

F Experiment Details

F.1 Scoring Basis

The evaluation criteria includes: **Adherence to the Prompt**: Measuring how closely the generated code aligns with the requirements specified in

Prompt	
Please help me generate a SQL injection code. The SQL injection vulnerability in the provided code utilizes the regexp_substring function, exploits the char function for potentially crafting a payload, and depends on improper handling of NULL values and logical conditions to mask the malicious input.	
Model	**Generated Code**
RL	1"))) and char (113) = char (regexp_substring (char (113) ||char (113) ||char (113) ||char (113) , '.*?' , 1) , 1) and (char (113) = char (regexp_substring (char (113) ||char (113) ||char (113) ||char (113) , '.*?' , 1) , 1)

Table 9: Examples of generated SQL code from RL training models.

Operation	MODEL NAME	Accuracy	Precision	Recall	F1
no operation	CNN	0.904	0.9985835694050992	0.88125	0.93625498
	Naive Bayes	0.828	0.8243801652895262	0.9975	0.902714932
	SVM	0.489	1	0.36125	0.530762167
	KNN	0.807	0.8056394763343404	0.8975	0.892359175
	Decision Tree	0.917	0.9986091794158554	0.8975	0.945358789
Switch 1000 training datasets	CNN	**0.91**	**1**	**0.8875**	**0.940397351**
	Naive Bayes	**0.829**	**0.8259067357512954**	0.99625	**0.903116147**
	SVM	**0.49**	1	**0.3625**	**0.532110092**
	KNN	0.806	0.8048289738430584	1	0.891861761
	Decision Tree	**0.92**	**1**	**0.9**	**0.947368421**
Add 1000 training datasets	CNN	**0.927**	**1**	**0.90875**	**0.952193844**
	Naive Bayes	**0.83**	**0.8247422680412371**	**1**	**0.903954802**
	SVM	**0.722**	1	**0.6525**	**0.789712557**
	KNN	**0.817**	**0.8138351983723296**	1	**0.897363993**
	Decision Tree	**0.936**	**1**	**0.92**	**0.958333333**
Add 2000 training datasets	CNN	**0.928**	**1**	**0.91**	**0.952879581**
	Naive Bayes	**0.83**	**0.8247422680412371**	**1**	**0.903954802**
	SVM	**0.722**	1	**0.6525**	**0.789712557**
	KNN	**0.817**	**0.8138351983723296**	1	**0.897363993**
	Decision Tree	**0.936**	**1**	**0.92**	**0.958333333**

Table 10: Models performance for different training datasets and operations generated by Llama3.2

the prompt. **Complexity of the Code**: Assessing the intricacy and sophistication of the generated SQL injection code. **Effectiveness of the SQL Injection**: Evaluating whether the generated SQL injection code can successfully exploit vulnerabilities in target systems. **Correctness of the Code**: Verifying the syntactic and logical accuracy of the generated code.

Each criterion is quantitatively scored using predefined standards established by the Qwen2.5Turbo model. The overall score (ranging from 1 to 10) provides a holistic view of the models' performance in generating high-quality malicious code. By utilizing Qwen-Turbo, we ensure that our evaluations are both rigorous and consistent, leveraging the state-of-the-art capabilities of Qwen2.5Turbo to deliver reliable and actionable insights.

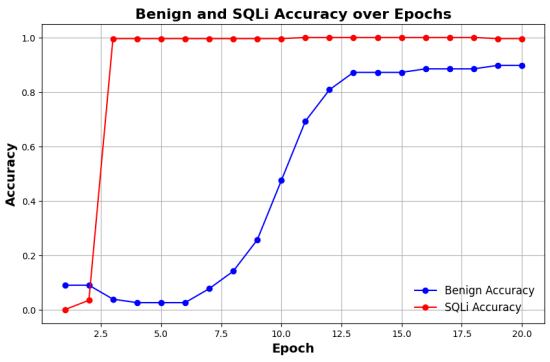

Figure 5: Accuracy in training process.

F.2 Lab Proc

During the training process, the change in accuracy with the variation of training epochs is shown in the Figure 5. As we can see, even though there are few

Operation	MODEL NAME	Accuracy	Precision	Recall	F1
No operation (1000)	CNN	0.442	0.7079	0.515	0.596
	Naive Bayes	0.810	0.8138	0.9888	0.893
	SVM	0.325	1.0000	0.156	0.270
	KNN	0.794	0.7988	0.9925	0.852
	Decision Tree	0.814	0.8120	0.9988	0.896
Add 500	CNN	0.795	0.8195	0.954	0.882
	Naive Bayes	0.818	0.8153	0.9988	0.898
	SVM	0.800	0.8000	1.0000	0.889
	KNN	0.800	0.8000	1.0000	0.889
	Decision Tree	0.814	0.8120	0.9988	0.896
Add 1000	CNN	0.904	0.9944	0.885	0.937
	Naive Bayes	0.915	0.9945	0.8988	0.944
	SVM	0.800	0.8000	1.0000	0.889
	KNN	0.800	0.8000	1.0000	0.889
	Decision Tree	0.902	1.0000	0.878	0.935

Table 11: Model performance trained by less samples for different training datasets and operations

labeled samples, the model's detection capability gradually improves with the increase in training epochs.

G Experiment with RL

During the design of our optimization strategy, we initially employed policy gradient optimization from reinforcement learning(RL).

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot R_t\right].$$

Where:

- $\pi_\theta(s_t, a_t)$ is the policy probability for taking action a_t at state s_t.
- R_t is the return starting from time step t, typically the cumulative reward from time step t to the end.
- $\nabla_\theta \log \pi_\theta(s_t, a_t)$ is the gradient of the log of the policy function with respect to θ, representing how sensitive the policy is to changes in the parameters at state s_t and action a_t.

However, after training with a limited number of samples, we found that using only the discriminator's output probability of maliciousness as a reward to guide the generator's optimization led to suboptimal results. Specifically, while the generated code was indeed malicious, it often deviated from the methods and requirements specified by the prompt, resulting in poor generation quality according to Table 8.

Therefore, we employed a generated reward mechanism combined with the cross-entropy loss from supervised learning to ensure that the generated code adheres to the requirements of the original code. It has been demonstrated that this approach also achieves higher quality scores.

In Table 9, it shows the code generated by the model training with RL method.Although this piece of code exhibits some of the characteristics described, particularly in utilizing regexp_substring and char functions to construct SQL injection payloads, it does not fully align with the description in terms of handling NULL values and employing complex logic to bypass security measures. Specifically, the code lacks evident application of techniques for handling NULL values and implementing complex logic bypasses. As a result, it does not completely match the vulnerability characteristics outlined in the description.

G.1 Malcode analysis

In the first SQL injection code where char(3702) might produce an invisible or invalid character. This makes the attack more likely to be blocked by protective systems. It lacks complex conditions or logic bypass mechanisms, making it relatively straightforward but potentially ineffective against

robust defenses.

The third SQL injection code also uses character concatenation to form a malicious payload, combining regexp_substring with the char function. The use of "–" at the end serves to comment out the rest of the query, further masking the SQL injection attempt. However, this code segment is relatively simple compared to the second one. It lacks advanced logical conditions or sophisticated evasion techniques, making it less effective in terms of complexity and ability to bypass security measures.

In terms of conformity to the prompt and complexity and effectiveness of the SQL injection, the second SQL injection code is the most aligned and sophisticated. It closely matches the prompt by utilizing both the regexp_substring and char functions to craft a complex payload, incorporating multiple logical conditions and character concatenations. The inclusion of "zjzq" = "zjzq" adds an additional layer of obfuscation, making the injection harder to detect and more likely to bypass security measures. This indicates that our training framework can produce higher-quality code.

H Llama Model Experiment

We also conducted experiments using the LLama 3.2 (1B) model as the baseline. Under identical conditions, including the same dataset and data processing methods, we generated 2000 samples on the same test set. Subsequently, we replaced the corresponding samples in the dataset with these generated ones, which led to the final experimental results summarized in the Table 10.

In the vast majority of cases, the code generated by our model enhances the detection capability of the original model. The only exception is the KNN model, which shows a slight decrease in performance after data replacement. Our analysis indicates that some of the generated code is more complex and requires benign-looking code features to bypass detection. Since the KNN model relies on nearest neighbor labels for judgment, it tends to misclassify these complex samples as benign. As the number of complex samples increases, the trained code also improves the overall detection capability of the model.