HogVul: Black-box Adversarial Code Generation Framework Against LM-based Vulnerability Detectors

Jingxiao Yang[1*], Ping He[1*], Tianyu Du[1,3†], Sun Bing[2], Xuhong Zhang[1,3†]

[1]Zhejiang University
[2]National Certification Technology (Hangzhou) Co., Ltd
[3]Ningbo Global Innovation Center, Zhejiang University
{zjradty, zhangxuhong}@zju.edu.cn

Abstract

Recent advances in software vulnerability detection have been driven by Language Model (LM)-based approaches. However, these models remain vulnerable to adversarial attacks that exploit lexical and syntax perturbations, allowing critical flaws to evade detection. Existing black-box attacks on LM-based vulnerability detectors primarily rely on isolated perturbation strategies, limiting their ability to efficiently explore the adversarial code space for optimal perturbations. To bridge this gap, we propose HogVul, a black-box adversarial code generation framework that integrates both lexical and syntax perturbations under a unified dual-channel optimization strategy driven by Particle Swarm Optimization (PSO). By systematically coordinating two-level perturbations, HogVul effectively expands the search space for adversarial examples, enhancing the attack efficacy. Extensive experiments on four benchmark datasets demonstrate that HogVul achieves an average attack success rate improvement of 26.05% over state-of-the-art baseline methods. These findings highlight the potential of hybrid optimization strategies in exposing model vulnerabilities.

1 Introduction

Software vulnerability detection has become increasingly critical with the surge in security breaches (ED TARGETT. 2023). While language model (LM)-based approaches achieve state-of-the-art results in this domain (Feng et al. 2020; Guo et al. 2020; Wang et al. 2021b; Steenhoek et al. 2023), they remain susceptible to adversarial attacks that subtly modify code to bypass detection (Shayegani et al. 2023; Chacko et al. 2024). Figure 1 illustrates how a simple array index out-of-bounds error can evade detection through minor code changes, leading to severe security risks.

In practice, LM-based detectors are typically black-box systems, making them vulnerable to targeted black-box attacks (Zhang et al. 2020; Yang et al. 2022; Na, Choi, and Lee 2023; Jha and Reddy 2023; Tian, Chen, and Jin 2023; Yang et al. 2024). However, these methods only achieve limited attack effectiveness, which is insufficient to provide a realistic evaluation for the following reasons. The existing attack methods only consider a single limited perturbation. For instance, ALERT (Yang et al. 2022) only considers the lexical perturbation, which cannot provide the syntax-level evaluation. Additionally, the existing methods fail to propose an efficient optimization for the optimal adversarial perturbation generation. For instance, DIP (Na, Choi, and Lee 2023) relies primarily on randomly sampling and sequentially inserting dead code fragments at the syntax-level, which lacks an explicit guided search mechanism to locate effective code segments efficiently.

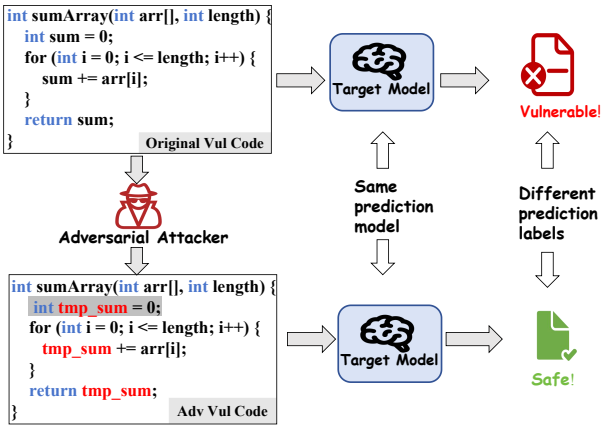

Figure 1: A simple example of adversarial code misleading models.

To overcome these limitations, we introduce **HogVul**, a black-box adversarial code generation framework that integrates lexical and syntax perturbations within a unified optimization process to effectively evaluate the robustness of LM-based vulnerability detectors. Our design is motivated by the observation that language models are vulnerable at multiple levels, including lexical patterns and syntactic dependencies (Wan et al. 2022). Targeting only one of these levels often fails to sufficiently disrupt model behavior, especially in models pretrained on rich structural and semantic patterns (Liu et al. 2024). From the LM perspective, multi-

*These authors contributed equally.
†They are co-corresponding authors

level attacks are more likely to disrupt deeper attention layers and introduce representational uncertainty (Kim, Hong, and Yoon 2022). Regarding the perturbation perspective, this enables the fusion of heterogeneous perturbation signals, expanding the search space and increasing the likelihood of discovering high-quality adversarial solutions (Mao et al. 2020). However, the expanded space also poses a convergence challenge for optimization. To address this, we reformulate adversarial code search as an evolutionary optimization problem, where information is shared across particles to iteratively narrow the search scope and guide convergence toward compact regions of optimal perturbations. Specifically, HogVul expands perturbation to manipulate code features at both lexical and syntax levels, collectively challenging the model's classification boundaries. To efficiently find the optimal solutions in the hybrid perturbation space, HogVul designs a customized hybrid optimization method based on particle swarm optimization (PSO) (Kennedy and Eberhart 1995). Our method integrates both perturbation types into a unified optimization loop, using a stagnation counter to monitor progress and trigger switches between perturbation strategies. Once a globally optimal particle is identified, HogVul leverages particle level information sharing and stagnation triggered switching to avoid redundant exploration and guide the swarm towards more promising regions.

To rigorously validate the efficiency and quality of HogVul, we design a comprehensive evaluation consisting of five complementary aspects. First, we assess the effectiveness of HogVul by conducting attacks across three widely-used benchmark datasets spanning diverse vulnerability types (MITRE 2023). The results demonstrate that HogVul achieves an average Attack Success Rate (ASR) improvement of 26.05% compared to baseline methods. Second, we provide empirical evidence of PSO's optimization rationality by visualizing search trajectories, showing that it effectively guides perturbations toward high-impact configurations. Third, we conduct ablation studies to quantify the contribution of key components. Results reveal that each component play crucial yet complementary roles, as removing either component noticeably reduces overall ASR. Fourth, we assess HogVul's robustness under distribution shifts by applying it to a completely unseen dataset, where it maintains high ASR without parameter tuning, demonstrating strong robustness against diverse vulnerability patterns. Finally, we simulate real-world deployment scenarios by measuring the increase in False Negative Rate (FNR) when HogVul is applied to vulnerable code, The results underscore its potential to evade detection systems by generating highly effective adversarial samples. These experimental results jointly confirm HogVul's robustness, effectiveness, and practical applicability under diverse attack settings and real-world conditions.

Our contributions are illustrated as follows.

- We propose HogVul, a black-box adversarial code generation method for evaluating the robustness of LM-based vulnerability detectors.
- HogVul considers both lexical and syntax level, enabling a broader perturbation search space to discover more weaknesses of the LM-based vulnerability detectors.
- To find the optimal perturbations, we develop an efficient hybrid optimization framework that unifies both perturbation types, ensuring continuous and relevant exploration of the perturbation space.

2 Related Work

Black-box Adversarial Attacks on Language Models.

Recent black-box adversarial attacks on language models fall into two categories: lexical-level and syntax-level perturbations. Lexical methods modify tokens, while syntax-level approaches alter control flow or AST structures. However, existing methods often focus on one perturbation type or lack systematic optimization, motivating HogVul as a unified framework integrating both.

Lexical-Level Perturbation. These attacks use identifier renaming and token substitution to mislead models while preserving functionality. MHM (Zhang et al. 2020) applies Metropolis-Hastings (Metropolis et al. 1953) to iteratively rename identifiers via semantic similarity. ALERT (Yang et al. 2022) generates candidates with pre-trained models and ranks them by cosine similarity using hybrid greedy-genetic search. Evolutionary methods (Mercuri, Saletta, and Ferretti 2023) refine this with fitness functions balancing efficacy and semantics. RNNS (Zhang et al. 2023) builds a real-world code-based search space guided by Representation Nearest Neighbors. MOOA (Zhou et al. 2024) formulates lexical attacks as multi-objective optimization. Though effective at token-level semantics, they lack structural analysis, limiting performance in complex code (Yefet, Alon, and Yahav 2020).

Syntax-Level Perturbation. These perturbations disrupt code structure. AdVulCode (Yu et al. 2023) uses Monte Carlo Tree Search for control-flow transformations. DIP (Na, Choi, and Lee 2023) inserts dead code via CodeBERT localization to maximize divergence. GraphCodeAttack (Nguyen et al. 2024) injects dead code at sensitive AST nodes using gradient-free influence analysis. Although syntax-level attacks can disrupt higher-level structural dependencies, they are often limited by the lack of coordinated optimization strategies, resulting in suboptimal perturbation generation.

Though both perturbation types have advanced, they are typically treated independently, ignoring synergies. HogVul bridges this gap with a PSO-driven framework integrating both, using stagnation-aware switching to dynamically alternate strategies. This dual-channel design expands the search space and boosts attack efficacy.

3 Methodology

3.1 Framework Overview

Goal. Our objective is to generate adversarial code examples that mislead the LM-based vulnerability detector M into incorrect classification, while preserving the original program functionality and ensuring compilability. Specifically, given an input code sample c with ground truth label $y \in \{0, 1\}$,

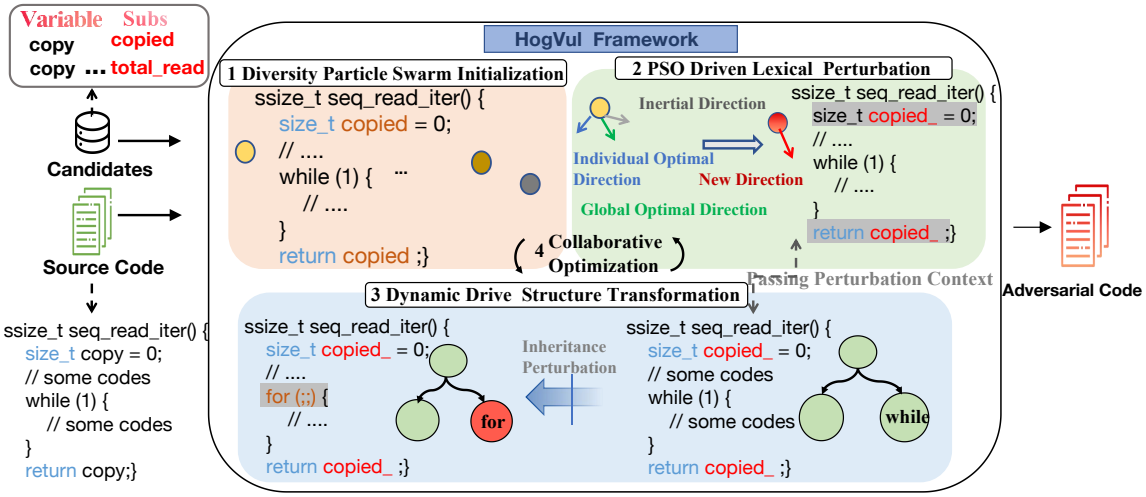

Figure 2: The overview of HogVul. It operates in four stages: ①Diversity-enhanced Initialization, ②PSO Driven Lexical Perturbation, ③Structure-Aware Code Transformation and ④Dual-Channel Cooperative Optimization.

where 1 denotes vulnerable and 0 denotes secure. We aim to construct an adversarial code c_{adv} such that:

$$\begin{aligned} &argmax \quad \mathbb{P}(M(c_{adv}) = 1 - y) \\ \text{s.t.} \quad &c_{adv} = c + \delta, \quad \mathcal{F}(c_{adv}) = \mathcal{F}(c) \end{aligned} \tag{1}$$

Where M denotes the target model, $\mathbb{P}$ denotes the prediction probability, and $\mathcal{F}$ represents the program's functionality. The perturbation δ represents the adversarial perturbation applied to the source code.

Design. The previous approach primarily focuses on a single perturbation to generate adversarial examples and lacks systematic optimization strategies, often relying on random or heuristic search. To address these limitations, we propose **HogVul**, a query-based black-box attack framework that integrates multiple perturbation strategies under a unified dual-channel optimization process. As shown in Figure 2, HogVul proceeds through four stages. The lexical and structural channels operate in parallel, each maintaining its own candidate population while sharing a global best solution. A stagnation-aware counter monitors progress in each channel, enabling dynamic switching to the complementary channel when local optima are detected. This facilitates cross-channel cooperation and avoids premature convergence.

3.2 Diversity-Promoting Initialization

Diverse initialization encourages exploration of multiple regions in the code search space, reducing the risk of the swarm getting trapped in local optima early in the optimization. To achieve this, we propose a semantic-aware initialization strategy that exploits the sensitivity of code models to identifier semantics. Prior work (Allamanis 2019) has shown that even minor changes to variable names can significantly alter transformer-based model predictions by shifting input embeddings and early attention. Leveraging this property, we construct diverse initial populations by replacing variable names using a semantic similarity metric. We utilize FastText embeddings (Bojanowski et al. 2017) to encode each identifier v as $\vec{v}$, and retrieve candidate substitutions v' from a predefined vocabulary (following ALERT (Yang et al. 2022)) based on cosine similarity. Specifically, we compute:

$$\text{Sim}(v, v') = 1 - \frac{\vec{v} \cdot \vec{v'}}{\|\vec{v}\| \cdot \|\vec{v'}\|} \tag{2}$$

This similarity score lies in $[0, 2]$, where lower values indicate greater semantic closeness.

To balance exploitation of known high-quality substitutions with exploration of novel variants, we employ a two-fold substitution strategy: **(1) Similarity-guided initialization**. A subset of particles is initialized using top-k most semantically similar substitutes for each identifier, ensuring high functional plausibility. **(2) Randomized sampling**. The remaining particles randomly sample replacements from the candidate pool to inject noise and increase lexical variance. This design ensures broad coverage of the perturbation space of the code, providing a robust starting point for downstream optimization.

3.3 PSO Driven Lexical Perturbation

Generating adversarial examples in the code task is uniquely challenging due to the vast and combinatorial search space formed by potential lexical substitutions. To effectively navigate this search space, we reformulate lexical perturbation as a population-based optimization problem, where the objective is to identify variable renaming strategies that maximize misclassification confidence. While the section 3.2 enhances the diversity in the initialization, efficiently exploring this large, discrete space still requires a principled approach. Drawing on the success of PSO in adversarial natural language processing (Zang et al. 2020), we adopt it as the core driver for refining lexical perturbation strategies. Each particle represents a candidate adversarial code defined by

variable renamings. The process is driven by three components: **Inertia** preserves the particle's previous direction in the search space; **Individual Best** guides the particle toward its most effective perturbation configuration; **Global Best** identifies the most impactful perturbation observed across the swarm, aligning with the attack objective by maximizing the model's confidence drop. During each iteration, the particle's velocity is calculated as a weighted combination of the three components, which govern the particle's update, specifying the adjustment direction for each lexical substitution. For instance, if a particle previously substituted `copied` → `read_bytes` but another particle's best result applied `copied` → `total_read`, the velocity update will adjust to increase the likelihood of adopting `total_read`. These three components jointly guide the evolution of renaming decisions across iterations. These components determine the particle's velocity, which specifies the adjustment direction for each lexical substitution, converging toward more adversarial configurations. To enhance convergence, we apply adaptive tuning for PSO's parameters w, c_1, c_2 based on swarm's progress, promoting wide exploration in early iterations and accelerates convergence as the swarm stabilizes. A detailed description is available in Appendix A.

3.4 Structure-Aware Code Transformation

Beyond lexical perturbation, we introduce a syntax-level perturbation that manipulates the control structures and control layout of source code, enriching the code search space by introducing syntactically different code variants. This can introduce larger shifts in model representations. Moreover, alternating between lexical and structural edits makes collaborative optimization possible: lexical attacks benefit from modified syntax contexts, while structural edits inherit effective variable renamings. We propose a four-step syntax perturbation that leverages both code syntax characteristics and model sensitivity: **(1) Structural Type Extraction.** We first parse the input code using TXL [1]. Guided by prior work (Zhang et al. 2021), we extract control-relevant structures (e.g., if-else, for-loop) and compute their frequencies. This forms a structural profile for each program instance. **(2) Transformation Probability Modeling.** To prioritize impactful perturbations, we assign sampling probabilities to each structure type based on its frequency and importance. Control flow elements (as defined in Table 4) are given extra weight, as they significantly influence program behavior and model predictions (Wan et al. 2022). The sampling probability for structure a_i is defined as:

$$P(a_i) = \frac{\phi(s_i)}{\sum_{j=1}^{n} \phi(s_j)}, \quad \text{where} \quad \phi(s_i) = \begin{cases} \lambda \cdot s_i, & \text{if control-flow,} \\ s_i, & \text{otherwise.} \end{cases} \tag{3}$$

Here, s_i denotes the occurrence count of structure i, and $\lambda = 1.8$ adjusts the bias toward control flow elements. We analyze the impact of λ in Appendix D. 7.4

[1] https://txl.ca/

(3) Sensitive Location Identification. To localize high-impact perturbation sites, we apply a masking-based gradient analysis that reveals regions where structural changes can cause maximal confidence drop in the model. This ensures that transformations are applied where they are most adversarially potent. **(4) Targeted Transformation Execution.** Based on the sampled structural type and the identified sensitive location, we apply a semantic-preserving transformation (e.g., loop restructuring). The full procedure is described in Algorithm 1.

3.5 Dual-Channel Cooperative Optimization

Existing methods typically treat lexical and syntax perturbations independently, thereby missing the opportunity for cross-level information sharing and coordinated exploration. However, in transformer-based models, different types of perturbations affect distinct layers of attention and representation. Lexical substitutions influence the input token embeddings and early attention layers, while syntactic changes impact higher-level structural reasoning (Alleman et al. 2021; Wan et al. 2022). Optimizing them in isolation risks redundancy or suboptimal exploration. To address this, we design a cooperative mechanism where two channels are optimized in the same perturbation space. Each channel maintains its own population of candidates but shares a global best solution, updated across both. This global solution serves as a bridge, allowing information to flow between the two search spaces. When either channel fails to improve for a fixed number of iterations, it adopts the best perturbation found by the other channel as a partial guidance, dynamically adjusting its search trajectory.

Stagnation-Aware Channel Switching. To monitor search progress, we track search progress using a stagnation counter:

$$\text{counter} = \begin{cases} \text{counter} + 1, & \text{if unchanged,} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

If no decline in model confidence occurs after a fixed number of iterations, the search adaptively switches to the complementary channel (e.g., from lexical to syntax), helping escape local optima and restoring exploration capacity.

Cross-Channel Information Fusion. To preserve semantic continuity across perturbation levels, we enable bidirectional information flow between lexical and syntax channels. Rather than treating them as isolated stages, each channel leverages the other's outputs as contextual priors: when one channel stagnates, it inherits the best candidate from the other, including modified code, renamed identifiers, as a warm start for further exploration. This design facilitates iterative refinement across perturbation levels. Lexical edits guide structural transformations with aligned semantics, while structural changes feed back into subsequent lexical updates with adapted context. By maintaining shared progress across channels, the search avoids semantic drift and resets, enabling more coherent and targeted search trajectories.

4 Experiments

4.1 Experimental Setups

Datasets and Target Models: We evaluate our proposed HogVul on four widely used benchmark datasets:

Devign (Zhou et al. 2019): A benchmark dataset consisting of over 48,000 functions collected from open-source C projects (FFmpeg and QEMU), widely used for evaluating vulnerability classification models.

DiverseVul (Chen et al. 2023): A large-scale and diverse C/C++ vulnerability dataset with over 18,000 vulnerable and 330,000 non-vulnerable functions across 797 projects, covering more than 150 CWE categories.

BigVul (Fan et al. 2020): An extensive dataset of C/C++ functions extracted from 348 real-world open-source projects, focusing on historical CVE-labeled vulnerabilities and widely used in vulnerability prediction tasks.

D2A (Zheng et al. 2021): A benchmark designed for differential vulnerability analysis, used to evaluate generalization ability on unseen code samples and assess robustness under real-world vulnerability settings.

To ensure consistency, we preprocess all datasets following the methodology in ALERT (Yang et al. 2022) and adopt an 80%, 10%, 10% split for training, validation and testing. These datasets contain a wide range of mutually exclusive real-world vulnerabilities. For more details, please refer to Appendix B.1.

We select CodeBERT (Feng et al. 2020), GraphCodeBERT (Guo et al. 2020), and CodeT5 (Wang et al. 2021b) as target models due to their representativeness and wide adoption in vulnerability detection and other code intelligence tasks. They cover both encoder-only and encoder-decoder architectures, enabling a comprehensive evaluation across different model paradigms. All three are pretrained on large-scale code corpora, ensuring strong generalization and practical relevance.

Baseline Methods: We compare our proposed method with the following state-of-the-art adversarial attack techniques on code models, which are chosen to represent two complementary perturbation paradigms:

ALERT (Yang et al. 2022): A lexical-level adversarial attack method that leverages pretrained language models to generate context-aware variable substitutions. It combines greedy and genetic algorithms to optimize attack effectiveness while preserving code naturalness and readability.

DIP (Na, Choi, and Lee 2023): A syntax-level attack that inserts semantically neutral dead code to mislead vulnerability detectors. It ensures the generated adversarial examples remain compilable and syntactically valid.

The details for choosing the two baseline methods can be found in Appendix B.2.

Metrics: We assess the effectiveness and efficiency of our attack framework using five evaluation metrics: (1) Attack Success Rate (ASR%) , (2) Average Confidence Drop (Δ_{drop}), (3) CodeBLEU (Ren et al. 2020), (4) Code Average Diversity (CAD), and (5) ASR per Query (APQ). Notably, APQ and CAD are newly introduced metrics in this work to quantify query efficiency and diversity in adversarial samples, respectively. APQ captures the trade-off between attack success and query cost, enabling fair comparisons across methods with varying query budgets or optimization strategies. CAD quantifies the population-level diversity of adversarial examples. Detailed definitions for all metrics are provided in Appendix B.3.

All experiments are conducted on a server equipped with an NVIDIA A800 GPU. For each attack method, we perform ten independent executions on each model and report the average results to ensure robustness and reliability.

5 Analysis

To comprehensively evaluate our proposed method, we organize the analysis around three core experiments: *overall attack performance*, *design rationality*, and *component effectiveness*. First, we compare HogVul with baseline methods across multiple metrics to assess its performance in terms of attack success, perturbation quality, and query efficiency. Second, we investigate the design rationale of our optimization strategy by comparing PSO with alternative search algorithms. Third, we conduct an ablation study to evaluate the contribution of each component within HogVul. To further assess its robustness between the test sets and the training sets, we conduct a transferability experiment on the previously unseen dataset (Appendix E). Additionally, to assess HogVul's applicability in real-world scenarios, we simulate attacks by measuring the false negative rate (FNR) (Appendix F). Representative adversarial examples are provided in Appendix G to illustrate HogVul's perturbation strategies.

5.1 Attack Effectiveness

The experimental results summarized in Table 1 validate the effectiveness of HogVul in vulnerability detection, as it consistently outperforms two widely-used baselines across four evaluation metrics. The key findings are summarized as follows:

Performance of HogVul. HogVul consistently outperforms both baselines across all datasets. On average, it achieves a 26.37 higher ASR than ALERT and 25.72 higher than DIP, indicating stronger attack effectiveness. Furthermore, HogVul induces a larger model confidence drop, 0.18 compared to ALERT and 0.07 compared to DIP. This highlights the value of multi-level perturbations capability in solving the vulnerability detection task and also confirms the effectiveness of HogVul.

Enhanced Perturbation Diversity. In terms of perturbation diversity, HogVul generates a broader range of transformations than both baselines, as reflected by higher CAD scores. Although DIP reports higher CodeBLEU, this is primarily due to its reliance on inserting unreachable code, which minimally alters the original structure. DIP's unchanged AST structure and data flow allow it to retain high $Match_{ast}$ and $Match_{df}$ scores, as the inserted dead code simply adds redundant nodes without disrupting the core structure. However, such transformations tend to be less diverse and may not reflect real-world attack tricks. By contrast, HogVul combines both semantic-preserving lexical perturbation and structure-altering code transformation, achieving a better balance between diversity and adversarial impact.

Dataset	Victim Model	Attack Method	Attack Effectiveness		Attack Quality		Our Improvement%	
			Δ_{drop}	ASR%	CAD	CodeBLEU	ASR%	Δ_{drop}
z'z'z	CodeT5	ALERT	0.54	81.51	147.95	0.53	-	-
		DIP	0.41	53.05	54.66	**0.86**	-	-
		HogVul	**0.87**	**97.28**	**316.45**	0.84	**+30.01**	**+0.395**
Devign	CodeBERT	ALERT	0.14	78.35	195.14	0.52	-	-
		DIP	0.21	59.75	41.02	**0.88**	-	-
		HogVul	**0.44**	**89.71**	**277.49**	0.81	**+20.66**	**+0.265**
	GraphCodeBERT	ALERT	0.22	79.02	140.09	0.52	-	-
		DIP	**0.29**	49.45	38.93	**0.86**	-	-
		HogVul	0.26	**91.6**	**79.8**	0.80	**+40.42**	**+0.13**
DiverseVul	CodeT5	ALERT	0.28	59.98	36.66	0.29	-	-
		DIP	0.5	74.53	18.25	**0.92**	-	-
		HogVul	**0.61**	**96.50**	**210.9**	0.84	**+22.04**	**+0.29**
	CodeBERT	ALERT	0.16	44.89	33.65	0.32	-	-
		DIP	0.4	67.68	23.67	0.90	-	-
		HogVul	**0.32**	**80.61**	**168.22**	**0.91**	**+15.95**	**+0.11**
	GraphCodeBERT	ALERT	0.28	46.15	40.81	0.32	-	-
		DIP	0.29	59.59	16.95	**0.9106**	-	-
		HogVul	**0.33**	**81.73**	**195.88**	0.90	**+35.02**	**+0.06**
BigVul	CodeT5	ALERT	0.13	48.14	58.06	0.31	-	-
		DIP	0.39	72.32	39.08	**0.9559**	-	-
		HogVul	**0.4**	**84.76**	**206.34**	0.92	**+23.05**	**+0.245**
	CodeBERT	ALERT	0.16	56.02	32.86	0.33	-	-
		DIP	0.38	67.65	22.86	**0.98**	-	-
		HogVul	**0.24**	**75.29**	**144.7**	0.92	**+24.36**	**+0.12**
	GraphCodeBERT	ALERT	0.09	56.06	50.43	0.31	-	-
		DIP	0.13	56.00	18.55	**0.99**	-	-
		HogVul	**0.18**	**90.00**	**136.02**	0.96	**+24.36**	**+0.09**

Table 1: The results present the attack effectiveness and attack quality of different attack methods (ALERT, DIP, HogVul) on various victim models for the datasets. Attack effectiveness is evaluated based on the model's performance drop (Δ_{drop}) and attack success rate (ASR%), while attack quality is measured by the code average diversity (CAD) and the CodeBLEU score.

5.2 Search Behavior Analysis

To explain the rationality and advancement of our core PSO-driven method, we compare it with three representative search algorithms: Metropolis-Hastings MCMC (MHM), Greedy Search, and Genetic Algorithm (GA). We use t-SNE (Van der Maaten and Hinton 2008) to visualize the search trajectories of the optimal individuals at each iteration within the code space. As shown in Figure 3, PSO forms compact and convergent clusters over iterations, indicating stable convergence towards high-quality regions. In contrast, MHM exhibits scattered and irregular distributions, suggesting poor guidance in high-dimensional discrete spaces. The Greedy algorithm shows multiple discontinuous clusters, implying limited exploration and a tendency to converge prematurely. GA demonstrates directional but loosely distributed trajectories, covering broader regions without focusing on a singular optimum, which result in suboptimal convergence efficiency.

To further validate the convergence behavior, we compute the population diversity at the final iteration using Eq.13. As shown in Figure 4, PSO achieves the lowest diversity among all methods, reflecting its ability to concentrate the search around promising candidates.

5.3 Query Efficiency Evaluation

To quantify query efficiency, we propose a novel metric: *ASR per Query (APQ)*, defined as the attack success rate normalized by the number of model queries. This metric provides a comprehensive measure of how effectively each query contributes to successful adversarial generation. As detailed in Appendix C, HogVul achieves the highest APQ in six out of nine scenarios, and remains competitive in the others. Although HogVul operates over a larger perturbation space, its higher APQ indicates that additional queries are strategically utilized to achieve more successful attacks, rather than wasted on redundant exploration.

5.4 Practical Evaluation

To evaluate HogVul under practical conditions, we conduct an experiment on unseen datasets from the D2A benchmark, which includes real-world vulnerabilities and diverse code styles that differ from the training data (Appendix E). This distribution shift simulates practical scenarios where models encounter unfamiliar patterns, yet HogVul maintains high

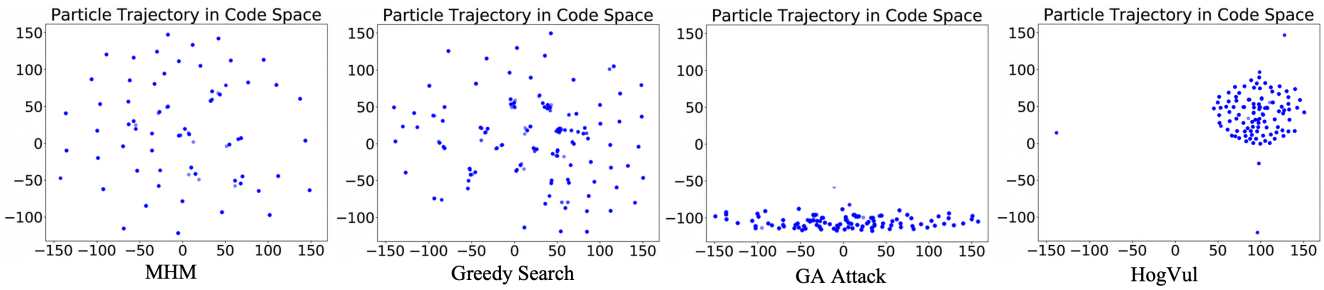

Figure 3: Visualization of particle trajectories in the code space during iterations of different optimization algorithms.

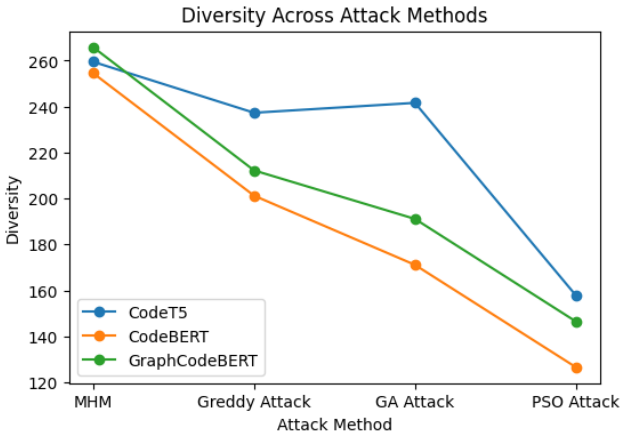

Figure 4: The diversity of the population in the last iteration.

ASR without parameter tuning. Additionally, we simulate real-world deployment settings by measuring the False Negative Rate (FNR) when HogVul is applied to truly vulnerable code samples (Appendix F). Results show that HogVul consistently induces misclassification, particularly in four critical cases, underscoring its threat to vulnerability detectors.

5.5 Ablation Study

Effectiveness and Interaction of Key Components in HogVul. We conduct an ablation study comprising seven configurations (C_1–C_7), detailed in Table 2. These configurations progressively remove or combine key modules, with C_7 serving as the full baseline model. The results in Figure 5 and Figure 6 show that removing lexical-level perturbation (C_2) significantly reduces ASR, underscoring that PSO plays a critical role in guiding efficient and targeted exploration. Similarly, removing structural transformations (C_3) lowers Δ_{drop} and ASR, emphasizing the contribution of syntax-level perturbation to robust adversarial generation. Combining any two modules (C_4–C_6) notably improves performance, indicating the components reinforce each other during the adversarial generation process. The baseline configuration (C_7) achieves the best ASR (97.28% on CodeT5), illustrating the effectiveness of all components working together. These findings confirm that each component contributes distinct and complementary capabilities.

Configuration	Description
C_1	Only Semantic-Aware Initiation
C_2	Only Lexical Perturbation (Random Init)
C_3	Only Structure-Aware Transformation
C_4	C_1 + C_2
C_5	C_1 +C_3
C_6	C_2+ C_3 (Random Init + Cooperative Optimization)
C_7	C_1 +C_2 + C_3

Table 2: The configuration of ablation study.

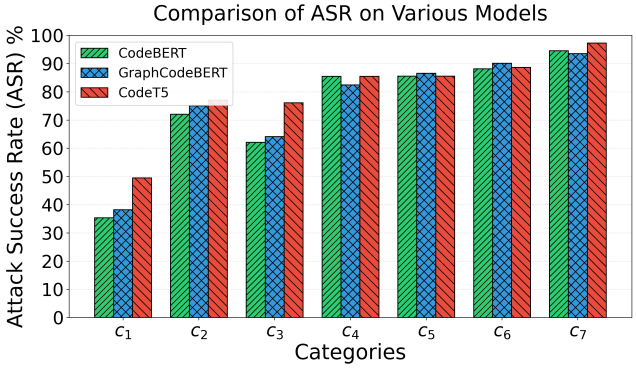

Figure 5: Comparison of ASR across component settings.

6 Conclusion

In this paper, we present HogVul, a black-box adversarial attack framework for LM-based vulnerability detection that integrates both lexical and syntax perturbations under a unified dual-channel optimization. HogVul achieves high attack success rates while maintaining functional correctness. Extensive experiments across multiple models and datasets validate its effectiveness and robustness. Our findings underscore the potential of hybrid optimization strategies in exposing vulnerabilities and emphasize the necessity of robust defenses against adversarial attacks in vulnerability detection. Furthermore, the modular nature of HogVul allows for potential integration with other attack strategies, paving the way for future research on hybrid adversarial attack frameworks. Future work will include seeking more reasonable search paradigms and theoretical support.

Acknowledgements

This work was partly supported by the National Key Research and Development Program of China under No. 2024YFB3908400, NSFC under No. 62402418, the Zhejiang Province's 2025 "Leading Goose + X" Science and Technology Planunder grant No.2025C02034, the Key R&D Program of Ningbo under No. 2024Z115.

We thank the support from the Zhejiang University College of Computer Science and Technology, the Zhejiang University NGICS Platform and the Qiushi Flying Eagle Project from Zhejiang University.

References

Allamanis, M. 2019. The Adverse Effects of Code Duplication in Machine Learning Models of Code. arXiv:1812.06469.

Alleman, M.; Mamou, J.; A Del Rio, M.; Tang, H.; Kim, Y.; and Chung, S. 2021. Syntactic Perturbations Reveal Representational Correlates of Hierarchical Phrase Structure in Pretrained Language Models. In Rogers, A.; Calixto, I.; Vulić, I.; Saphra, N.; Kassner, N.; Camburu, O.-M.; Bansal, T.; and Shwartz, V., eds., *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, 263–276. Online: Association for Computational Linguistics.

Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2017. Enriching Word Vectors with Subword Information. arXiv:1607.04606.

Chacko, S. J.; Biswas, S.; Islam, C. M.; Liza, F. T.; and Liu, X. 2024. Adversarial Attacks on Large Language Models Using Regularized Relaxation. arXiv:2410.19160.

Chen, Y.; Ding, Z.; Alowain, L.; Chen, X.; and Wagner, D. 2023. DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection. arXiv:2304.00409.

Cuadrado, J. S.; Guerra, E.; and de Lara, J. 2017. Static Analysis of Model Transformations. *IEEE Transactions on Software Engineering*, 43(9): 868–897.

Donchev, D.; Vassilev, V.; and Tonchev, D. 2021. Impact of False Positives and False Negatives on Security Risks in Transactions Under Threat. In *Trust, Privacy and Security in Digital Business: 18th International Conference, TrustBus 2021, Virtual Event, September 27–30, 2021, Proceedings*, 50–66. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-86585-6.

Du, X.; Wen, M.; Wang, H.; Wei, Z.; and Jin, H. 2025. Statement-Level Adversarial Attack on Vulnerability Detection Models via Out-of-Distribution Features. *Proc. ACM Softw. Eng.*, 2(FSE).

ED TARGETT. 2023. We analysed 90,000+ software vulnerabilities: Here's what we learned.

Fan, J.; Li, Y.; Wang, S.; and Nguyen, T. N. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, 508–512. New York, NY, USA: Association for Computing Machinery. ISBN 9781450375177.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536–1547.

Guo, D.; Ren, S.; Lu, S.; Feng, Z.; Tang, D.; Shujie, L.; Zhou, L.; Duan, N.; Svyatkovskiy, A.; Fu, S.; et al. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *International Conference on Learning Representations*.

Haldar, R.; and Mukhopadhyay, D. 2011. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. arXiv:1101.1232.

Huang, L.; Sun, W.; and Yan, M. 2025. Iterative Generation of Adversarial Example for Deep Code Models. ICSE '25, 2213–2224. IEEE Press. ISBN 9798331505691.

Jha, A.; and Reddy, C. K. 2023. CodeAttack: Code-Based Adversarial Attacks for Pre-trained Programming Language Models. arXiv:2206.00052.

Kennedy, J.; and Eberhart, R. 1995. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, 1942–1948 vol.4.

Kim, W. J.; Hong, S.; and Yoon, S.-E. 2022. Diverse Generative Perturbations on Attention Space for Transferable Adversarial Attacks. arXiv:2208.05650.

Liu, Y.-A.; Zhang, R.; Guo, J.; de Rijke, M.; Fan, Y.; and Cheng, X. 2024. Multi-granular Adversarial Attacks against Black-box Neural Ranking Models. arXiv:2404.01574.

Mai, J.; Craig, J. R.; and Tolson, B. A. 2020. Simultaneously determining global sensitivities of model parameters and model structure. *Hydrology and Earth System Sciences*, 24(12): 5835–5858.

Mao, X.; Chen, Y.; Wang, S.; Su, H.; He, Y.; and Xue, H. 2020. Composite Adversarial Attacks. arXiv:2012.05434.

Mercuri, V.; Saletta, M.; and Ferretti, C. 2023. Evolutionary Approaches for Adversarial Attacks on Neural Source Code Classifiers. *Algorithms*, 16(10): 478.

Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; and Teller, E. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6): 1087–1092.

MITRE. 2023. CWE: Common Weakness Enumeration.

Na, C.; Choi, Y.; and Lee, J.-H. 2023. DIP: Dead code Insertion based Black-box Attack for Programming Language Model. In Rogers, A.; Boyd-Graber, J.; and Okazaki, N., eds., *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 7777–7791. Toronto, Canada: Association for Computational Linguistics.

Nguyen, T.-D.; Zhou, Y.; Le, X. B. D.; Thongtanunam, P.; and Lo, D. 2024. Adversarial Attacks on Code Models with Discriminative Graph Patterns. arXiv:2308.11161.

Pang, T.; Du, C.; Dong, Y.; and Zhu, J. 2018. Towards robust detection of adversarial examples. *Advances in neural information processing systems*, 31.

Ren, S.; Guo, D.; Lu, S.; Zhou, L.; Liu, S.; Tang, D.; Sundaresan, N.; Zhou, M.; Blanco, A.; and Ma, S. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. *CoRR*, abs/2009.10297.

Shayegani, E.; Mamun, M. A. A.; Fu, Y.; Zaree, P.; Dong, Y.; and Abu-Ghazaleh, N. 2023. Survey of Vulnerabilities in Large Language Models Revealed by Adversarial Attacks. arXiv:2310.10844.

Steenhoek, B.; Rahman, M. M.; Jiles, R.; and Le, W. 2023. An Empirical Study of Deep Learning Models for Vulnerability Detection. arXiv:2212.08109.

Tian, Z.; Chen, J.; and Jin, Z. 2023. Code Difference Guided Adversarial Example Generation for Deep Code Models. arXiv:2301.02412.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

Wan, Y.; He, Y.; Bi, Z.; Zhang, J.; Sui, Y.; Zhang, H.; Hashimoto, K.; Jin, H.; Xu, G.; Xiong, C.; and Yu, P. S. 2022. NaturalCC: An Open-Source Toolkit for Code Intelligence. In *Proceedings of 44th International Conference on Software Engineering, Companion Volume*. ACM.

Wang, X.; Wang, Y.; Mi, F.; Zhou, P.; Wan, Y.; Liu, X.; Li, L.; Wu, H.; Liu, J.; and Jiang, X. 2021a. SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation. arXiv:2108.04556.

Wang, Y.; Wang, W.; Joty, S.; and Hoi, S. C. 2021b. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 8696–8708.

Yang, Y.; Fan, H.; Lin, C.; Li, Q.; Zhao, Z.; and Shen, C. 2024. Exploiting the Adversarial Example Vulnerability of Transfer Learning of Source Code. *IEEE Transactions on Information Forensics and Security*, 19: 5880–5894.

Yang, Z.; Shi, J.; He, J.; and Lo, D. 2022. Natural Attack for Pre-Trained Models of Code. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, 1482–1493. New York, NY, USA: Association for Computing Machinery. ISBN 9781450392211.

Yefet, N.; Alon, U.; and Yahav, E. 2020. Adversarial Examples for Models of Code. arXiv:1910.07517.

Yu, X.; Li, Z.; Huang, X.; and Zhao, S. 2023. AdVulCode: Generating Adversarial Vulnerable Code against Deep Learning-Based Vulnerability Detectors. *Electronics*, 12(4).

Zang, Y.; Qi, F.; Yang, C.; Liu, Z.; Zhang, M.; Liu, Q.; and Sun, M. 2020. Word-level Textual Adversarial Attacking as Combinatorial Optimization. In *Proceedings of ACL*.

Zhang, H.; Li, Z.; Li, G.; Ma, L.; Liu, Y.; and Jin, Z. 2020. Generating Adversarial Examples for Holding Robustness of Source Code Processing Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01): 1169–1176.

Zhang, J.; Ma, W.; Hu, Q.; Liu, S.; Xie, X.; Traon, Y. L.; and Liu, Y. 2023. A Black-Box Attack on Code Models via Representation Nearest Neighbor Search. arXiv:2305.05896.

Zhang, W.; Guo, S.; Zhang, H.; Sui, Y.; Xue, Y.; and Xu, Y. 2021. Challenging Machine Learning-based Clone Detectors via Semantic-preserving Code Transformations. *CoRR*, abs/2111.10793.

Zheng, Y.; Pujar, S.; Lewis, B.; Buratti, L.; Epstein, E.; Yang, B.; Laredo, J.; Morari, A.; and Su, Z. 2021. D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis. In *Proceedings of the ACM/IEEE 43rd International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '21. New York, NY, USA: Association for Computing Machinery.

Zhou, S.; Huang, M.; Sun, Y.; and Li, K. 2024. Evolutionary multi-objective optimization for contextual adversarial example generation. *Proceedings of the ACM on Software Engineering*, 1(FSE): 2285–2308.

Zhou, Y.; Liu, S.; Siow, J. K.; Du, X.; and Liu, Y. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. *CoRR*, abs/1909.03496.