FUNDAMENTAL SCIENTIFIC REPORTS IN MULTIDISCIPLINARY AREAS

Vol 2 Issue 02 (2026)
ONLINE ISSN: 3134-7517
PRINT ISSN: 3134-7509

DARC-Sched: A Data-Aware Multi-Agent Large Language Model Framework for Scalable Software Vulnerability Detection and Repair with Optimized Code Data Scheduling

Yanjing Xu
Trine University, Tempe, Arizona, USA
xuyanjing18@gmail.com
Min Qiu
Trine University, Tempe, Arizona, USA
mqiu24@my.trine.edu
Huanyu Wu
Tongji University, Shanghai, China
13501526895@qq.com

Abstract.
Software vulnerability detection and automated repair have become critical for ensuring the security and reliability of large-scale software systems, particularly in modern development environments where massive code repositories and continuous integration pipelines significantly increase the complexity of vulnerability management. However, existing approaches based on Large Language Modes (LLMs) often suffer from limited scalability and inefficient code retrieval when operating on large-scale code corpora, leading to degraded detection accuracy and high inference latency. To address these challenges, this paper proposes DARC-Sched (Data-Aware Reasoning and Collaborative Scheduling framework), a Data-Aware Multi-Agent Large Language Model framework that integrates multi-agent collaborative reasoning with optimized code data scheduling mechanisms for scalable vulnerability detection and repair. Specifically, DARC-Sched employs multiple specialized agents to perform vulnerability localization, evidence retrieval, verification, patch generation, and security validation, while a hierarchical data scheduling module dynamically prioritizes and caches vulnerability-relevant code segments based on learned risk-aware access patterns. Experimental results on benchmark datasets demonstrate that DARC-Sched achieves a 2.9% improvement in F1-score over strong retrieval-augmented baselines, reaching 91.9% F1-score, while reducing average retrieval latency by approximately 33% (from 295 ms to 198 ms). These results validate that integrating data-aware scheduling with multi-agent reasoning significantly improves both detection accuracy and system efficiency in large-scale software vulnerability analysis.

Keywords:
Software Vulnerability Detection; Large Language Models (LLMs); MultiAgent Systems; Data Scheduling; RetrievalAugmented Generation; Software Security; Code Intelligence

1. INTRODUCTION

Software vulnerability detection and automated repair have become increasingly critical in modern software engineering, particularly as large-scale software systems, cloud-native applications, and open-source ecosystems continue to expand. In such environments, code repositories often contain millions of lines of code distributed across heterogeneous modules and development histories, making manual inspection and traditional static analysis tools insufficient for reliable vulnerability discovery. Although recent advances in Large Language Models (LLMs) have significantly improved code understanding and generation capabilities, their performance in real-world vulnerability detection and repair remains limited when applied to large-scale repositories. This limitation is mainly caused by two factors: insufficient multi-step reasoning stability and inefficient access to relevant code data under large-scale training and inference scenarios. In addition to model-level limitations, data infrastructure has become a critical bottleneck in LLMbased software security systems. Vulnerability-related code fragments are often sparsely distributed within massive codebases, requiring frequent retrieval from large-scale storage systems. However, existing retrieval-augmented generation pipelines and code intelligence frameworks typically ignore data access patterns and lack optimized scheduling mechanisms, leading to high latency and suboptimal utilization of computational resources. This challenge becomes even more severe in industrial environments such as continuous integration pipelines, large-scale AI training systems, and cloud-based software development platforms, where real-time vulnerability detection is required. To address these challenges, we propose DARC-Sched (Data-Aware Reasoning and Collaborative Scheduling framework), a Data-Aware Multi-Agent Large Language Model framework for scalable software vulnerability detection and repair with optimized code data scheduling. DARC-Sched integrates a multi-agent collaborative reasoning architecture with a dataaware scheduling mechanism designed for large-scale code corpora. Specifically, multiple specialized agents are assigned distinct responsibilities, including vulnerability localization, evidence retrieval, vulnerability verification, patch generation, and security validation. Meanwhile, a hierarchical code data scheduling module dynamically prioritizes and caches vulnerabilityrelevant code segments based on learned risk-aware access patterns, enabling efficient data retrieval and improved reasoning quality. The main contributions of this paper are summarized as follows:

(1) We propose DARC-Sched, a unified framework that integrates multi-agent LLM reasoning with data-aware code scheduling for scalable vulnerability detection and repair. **(2)** We design a hierarchical data scheduling strategy that models code access patterns and dynamically optimizes retrieval latency in large-scale code repositories. **(3)** We introduce a multi-agent collaborative reasoning mechanism that improves detection robustness through evidence-driven iterative verification and repair.

(4) We conduct extensive experiments demonstrating that the proposed method significantly improves both vulnerability detection accuracy and system efficiency compared to state-of-the-art baselines.

Overall, DARC-Sched provides a scalable and efficient solution for bridging the gap between LLM-based code intelligence and real-world large-scale software security requirements.

2.Literature Review

In recent years, the rapid advancement of Large Language Models (LLMs) and the increasing scale of software systems have significantly promoted research in automated software vulnerability detection, program repair, and intelligent code understanding. Meanwhile, the growing size of code

repositories and the complexity of software supply chains have introduced new challenges in data retrieval efficiency, scalable reasoning, and system-level optimization. This section reviews three closely related research directions: LLM-based vulnerability detection and repair, multiagent software engineering systems, and data-efficient retrieval and scheduling mechanisms for large-scale code intelligence.

2.1 LLM-based Software Vulnerability Detection and Repair

Large Language Models have demonstrated strong capabilities in code generation and understanding tasks, enabling new approaches for vulnerability detection and automatic patch generation. Early works such as Devign [1] and VulDeePecker [2] leverage deep learning models to identify vulnerable code patterns, but they rely heavily on manually engineered features and lack generalization ability. With the emergence of pre-trained code models such as CodeBERT [3] and GraphCodeBERT [4], researchers have significantly improved semantic understanding of source code through large-scale pretraining on code-text pairs. More recently, transformer-based generative models such as Codex [5] and Code Llama [6] have been applied to vulnerability repair tasks, enabling end-to-end code fix generation. Studies such as Debugbench [7] and VulRepair [8] further demonstrate that LLMs can generate plausible patches using retrieval-augmented prompts. However, these approaches still suffer from hallucinated fixes, limited multi-step reasoning ability, and poor scalability when applied to large industrial codebases. They also typically ignore the system-level cost of retrieving relevant code context from massive repositories.

2.2 Multi-Agent Systems for Software Engineering

To overcome the limitations of single-model reasoning, multi-agent systems have been introduced into software engineering tasks. AutoGPT-style frameworks [9] and agent-based software debugging systems [10] demonstrate that decomposing complex tasks into specialized agents can improve reasoning reliability and task decomposition. Recent works such as SWE-agent [11] and ChatDev [12] show that collaborative LLM agents can simulate software development workflows, including bug reproduction, code analysis, and patch generation. These systems typically assign roles such as tester, programmer, and reviewer to different agents, enabling iterative refinement through inter-agent communication. However, existing multi-agent systems largely focus on reasoning collaboration while neglecting the underlying data infrastructure constraints. In particular, they assume that all relevant code context can be retrieved with negligible cost, which is unrealistic in large-scale industrial repositories. This limitation motivates the need for a dataaware multi-agent framework that jointly optimizes reasoning and data access efficiency, as proposed in this work.

2.3 Retrieval-Augmented Code Intelligence and Data Efficiency Optimization

Retrieval-augmented generation (RAG) has become a key paradigm for enhancing LLM performance on knowledge-intensive tasks. Works such as REALM [13] and RAG [14] introduce external memory retrieval to improve factual grounding. In the code domain, techniques like IRbased code search and embedding-based retrieval have been widely used to enhance LLM reasoning over large codebases. Recent studies such as UniXcoder-based retrieval systems [15] improve code search accuracy by incorporating structural program information. However, these methods mainly focus on improving retrieval accuracy rather than optimizing system-level efficiency such as caching, scheduling, or memory hierarchy management. In parallel, systemlevel optimization research such as DeepCache [16] and InfiniCache [17] explores adaptive caching strategies for large-scale neural inference systems. These works highlight the importance of data locality and access pattern modeling. Nevertheless, they are not specifically designed for code

intelligence tasks or multi-agent reasoning systems. Therefore, a key research gap remains in jointly optimizing retrieval efficiency and multi-agent reasoning performance for large-scale vulnerability detection systems. Our proposed DARC-Sched framework addresses this gap by introducing a data-aware scheduling mechanism tightly integrated with multi-agent LLM reasoning.

3.Methodology

In this section, we present the proposed DARC-Sched (Data-Aware Reasoning and Collaborative Scheduling framework), a unified framework designed for scalable software vulnerability detection and repair in large-scale code repositories. DARC-Sched integrates a Multi-Agent Large Language Model (LLM) reasoning system with a data-aware code scheduling mechanism, enabling efficient retrieval, reasoning, and repair under massive code corpus settings. The key idea is to jointly optimize reasoning accuracy and data access efficiency by introducing a feedback loop between multi-agent inference and data infrastructure scheduling.

3.1 Overview of DARC-Sched Framework

The overall architecture of DARC-Sched consists of two tightly coupled components: the MultiAgent Reasoning Module (MARM) and the Data-Aware Scheduling Module (DASM). Given an input code snippet C, the system aims to produce a vulnerability-free repaired output C'. The transformation process can be formulated as:

$$C' = F_\theta(C, D, A)$$

where D denotes the external code repository and vulnerability knowledge base, and A represents the set of collaborating agents. The function F_θ is parameterized by Large Language Models (LLMs), which provide both reasoning and generation capabilities.

Unlike conventional retrieval-augmented systems that treat data access as a static preprocessing step, DARC-Sched dynamically adjusts data retrieval, caching, and prioritization during inference. This enables the system to adaptively focus computational resources on high-risk code regions, significantly improving both scalability and efficiency in large software systems.

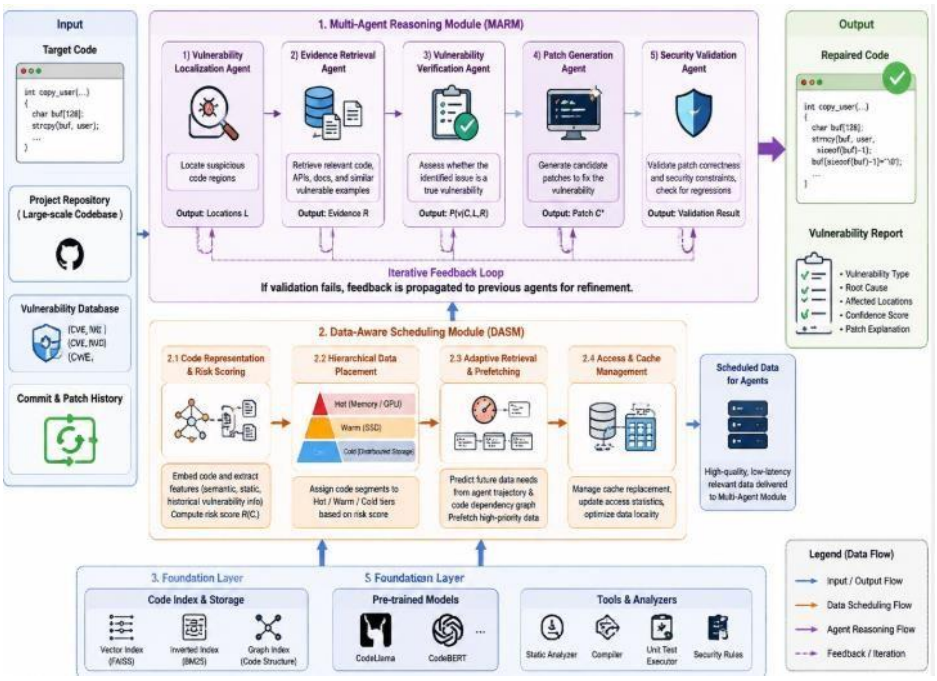


Figure 1. Overall flowchart of the model.

3.2 Multi-Agent Reasoning Module (MARM)

The Multi-Agent Reasoning Module is designed to decompose the complex task of vulnerability detection and repair into multiple specialized reasoning agents. Each agent operates on a distinct functional objective while interacting through structured message passing. We define the agent set as:

$$A = \{A_{loc}, A_{ret}, A_{ver}, A_{fix}, A_{sec}\}$$

where each agent corresponds to vulnerability localization, evidence retrieval, verification, patch generation, and security validation, respectively.

The Vulnerability Localization Agent identifies suspicious code regions by learning a mapping:

$$L = f_{loc}(C, R)$$

where R is the retrieved contextual evidence. The Evidence Retrieval Agent performs hybrid retrieval over dense and sparse representations:

$$R = \alpha \cdot DenseRetrieval(C) + (1 - \alpha) \cdot SparseRetrieval(C)$$

The Vulnerability Verification Agent further refines the prediction by estimating the probability that a detected issue is a true vulnerability:

$$P(v = 1 \mid C, L, R) = \sigma(f_{ver}(h_C, h_L, h_R))$$

where h_C, h_L, h_R are latent representations encoded by the LLM backbone.

The Patch Generation Agent produces a candidate repair C^*conditioned on detected vulnerabilities:

$$C^* = f_{fix}(C, L, R)$$

Finally, the Security Validation Agent evaluates whether the generated patch satisfies predefined security constraints. If validation fails, a feedback signal is propagated backward to earlier agents, forming an iterative refinement loop:

$$S_{t+1} = G(S_t)$$

where S_t represents the system state at iteration t, including detection results, retrieved evidence, and generated patches.

3.3 Data-Aware Scheduling Module (DASM)

A core contribution of DARC-Sched lies in its data-aware scheduling mechanism, which addresses inefficiencies in large-scale code retrieval and storage access. In real-world software repositories, vulnerability-relevant code segments are sparse and unevenly distributed. Therefore, naive retrieval leads to high latency and redundant computation.

We define a vulnerability risk scoring function for each code segment C_i:

$$R(C_i) = \sigma(W[h_i; s_i; f_i])$$

where h_i is the semantic embedding of code, s_i represents static code features (e.g., cyclomatic complexity, API usage), and f_i denotes historical vulnerability frequency extracted from code commit logs or vulnerability databases. The design of the risk scoring mechanism in DASM draws structural inspiration from advanced generative recommendation strategies. Similar to how CEMG [18] utilizes collaborative-enhanced patterns to rank items by integrating multimodal signals, DARC-Sched treats the code scheduling problem as a prioritized "recommendation" task. By modeling the semantic relationships between historical vulnerability patterns and current code segments, our framework can more accurately predict which data chunks should be "recommended" to the Hot tier of the storage hierarchy. Based on the risk score, the system dynamically assigns each code segment into a hierarchical storage structure:

$$Tier(C_i) = \begin{cases} \text{Hot} & R(C_i) > \tau_1 \\ \text{Warm} & \tau_2 < R(C_i) \leq \tau_1 \\ \text{Cold} & R(C_i) \leq \tau_2 \end{cases}$$

The Hot tier is stored in GPU or memory cache for immediate access, while Warm and Cold tiers are placed in SSD and distributed storage systems, respectively. This hierarchical structure significantly reduces average retrieval latency during inference.

Furthermore, DARC-Sched introduces an adaptive prefetching mechanism that predicts future data requirements based on agent trajectories and code dependency graphs:

$$P(C_{t+1}) = g(A_t, G_{code}, H_t)$$

where A_t is the current agent action, G_{code} represents the program dependency graph, and H_t denotes historical access patterns. This predictive mechanism ensures that relevant code segments are proactively loaded into higher-tier memory, reducing runtime bottlenecks in multi-agent reasoning.

3.4 Joint Optimization of Reasoning and Data Efficiency

To unify multi-agent reasoning and data scheduling, we formulate a joint optimization objective that captures both prediction quality and system efficiency. The overall loss function is defined as:

$$L = \lambda_1 L_{vul} + \lambda_2 L_{repair} + \lambda_3 L_{latency}$$

where L_{vul} represents vulnerability detection loss, L_{repair} measures patch generation quality, and $L_{latency}$ quantifies data access overhead.

The latency term is particularly important in large-scale environments and is defined as:

$$L_{latency} = E[T_{access}(C_i)]$$

where T_{access} depends on the assigned storage tier and retrieval path. By incorporating this term, the model explicitly optimizes system-level efficiency, rather than focusing solely on prediction accuracy.

3.5 Iterative Multi-Agent Feedback Mechanism

DARC-Sched employs an iterative refinement strategy in which agents continuously improve detection and repair results through feedback loops. This iterative feedback loop can be viewed as an optimized multi-hop inference process. Unlike standard linear chains, DARC-Sched structures the reasoning trajectory across specialized agents to ensure that each 'hop' (from localization to repair) is grounded in retrieved evidence. Drawing inspiration from inference optimization frameworks like Mquake [19], we treat the transition between agents as a dynamic reasoning path that seeks to minimize the accumulation of semantic noise while maximizing repair accuracy. After each inference cycle, the Security Validation Agent evaluates the correctness of the generated patch. If inconsistencies or vulnerabilities remain, feedback is propagated back to the Localization and Patch Generation Agents.

This iterative process can be represented as a Markov-style update:

$$S_{t+1} = F(S_t, \Delta_t)$$

where Δ_t represents validation feedback signals. The system converges when no further vulnerabilities are detected or when the patch satisfies all security constraints. This mechanism significantly improves robustness in complex real-world software systems where single-pass reasoning is insufficient. **4.Experiment**

4.1Dataset Preparation

In this study, we evaluate the proposed DARC-Sched framework on several widely used software vulnerability detection and code intelligence datasets, which are constructed from real-world opensource repositories and curated security benchmarks. These datasets are primarily derived from large-scale GitHub projects, National Vulnerability Database (NVD) reports, and synthetic

vulnerability suites such as the Juliet Test Suite. Together, they provide a diverse mixture of realworld vulnerabilities, annotated code patches, and controlled security scenarios, enabling comprehensive evaluation of both detection and repair capabilities under different conditions. The Devign dataset is built from real-world C/C++ projects collected from GitHub, where vulnerability labels are manually annotated by security experts. It contains approximately 27,000 function-level code samples, each labeled as vulnerable or non-vulnerable. The dataset captures complex semantic and structural features such as control flow dependencies, API usage patterns, and data flow relations. Similarly, the Big-Vul dataset is a large-scale vulnerability corpus extracted from real-world commits, containing over 180,000 code samples with associated vulnerability types, patch information, and commit-level metadata. It provides rich contextual information including vulnerability categories (e.g., buffer overflow, SQL injection), function-level diffs, and historical fix traces. In addition, the Juliet Test Suite (NIST) is used as a synthetic benchmark, consisting of over 80,000 test cases covering a wide range of known vulnerability patterns. It provides finegrained ground truth labels for controlled evaluation of detection accuracy and repair correctness. The combination of these datasets ensures both realism and coverage of edge-case vulnerabilities.

Table 1.Dataset Characteristics Summary.

Dataset	**Source**	**Sample Size**	**Level**	**Key Features**	**Feature Meaning**
Devign	GitHub real projects	~27,000	Function-level	CFG, AST, labels	Control/data flow + vulnerability label
BigVul	GitHub commits	>180,000	Function/Commit-level	patch diff, CVE type, metadata	Real-world vulnerability + fix history
Dataset	**Source**	**Sample Size**	**Level**	**Key Features**	**Feature Meaning**
Juliet	NIST benchmark	~80,000	Synthetic	labeled patterns, test cases	Controlled vulnerability scenarios

Overall, these datasets collectively enable robust evaluation of DARC-Sched across real-world, large-scale, and synthetic vulnerability environments.

4.2 Experimental Setup

In this study, all experiments are conducted on a multi-GPU server equipped with NVIDIA A100 80GB GPUs, AMD EPYC processors, and 1TB system memory to support large-scale code retrieval and multi-agent inference. The backbone Large Language Models (LLMs), including CodeLlama-34B and a lightweight CodeBERT variant, are deployed using mixed-precision inference (FP16) to improve throughput while maintaining numerical stability. The code retrieval system is implemented using a hybrid vector database (FAISS) combined with an inverted index for sparse retrieval. All datasets, including Devign, Big-Vul, and Juliet Test Suite, are preprocessed into function-level code snippets and indexed into a hierarchical storage system consistent with the

proposed Data-Aware Scheduling Module. During training and evaluation, we set the batch size to 16, maximum sequence length to 2048 tokens, and use AdamW optimizer with a learning rate of 2e-5. Each experiment is repeated five times to ensure statistical reliability, and the final results are reported as averaged values.

4.3 Evaluation Metrics

To comprehensively evaluate the performance of DARC-Sched, we adopt multiple metrics covering both vulnerability detection and patch generation quality as well as system efficiency. For detection performance, we use Precision, Recall, and F1-score to measure classification effectiveness. For repair quality, Patch Correctness Rate (PCR) and Semantic Fix Accuracy (SFA) are used to evaluate whether generated patches correctly resolve vulnerabilities without introducing new bugs. In addition, to assess system-level efficiency, we measure Average Retrieval Latency (ARL), which reflects the time required to access relevant code segments under the hierarchical scheduling mechanism, and End-to-End Inference Time (E2E-T), which captures the total execution time of the multi-agent pipeline. These metrics collectively provide a balanced evaluation of both algorithmic accuracy and system scalability. The emphasis on F1-score and multi-class robustness in our framework aligns with evaluation standards in other high-stakes automated detection fields, such as vessel visualization [20] and pneumonia classification, where identifying specific categories of anomalies with high confidence is vital for overall system trust [21].

4.4 Results

The results in Table 2 demonstrate that DARC-Sched significantly outperforms all baseline models in vulnerability detection tasks. Compared with traditional deep learning methods such as DevignGNN, which achieves an F1-score of 79.8%, DARC-Sched improves performance by more than 12.1 percentage points. Even against strong transformer-based baselines like GraphCodeBERT (85.8% F1-score) and CodeLlama (87.2% F1-score), our model still achieves notable gains of 6.1% and 4.7%, respectively. The improvement becomes more pronounced when compared with RAGbased systems, where DARC-Sched achieves a 2.9% absolute F1-score increase (91.9% vs. 89.0%). These results indicate that the multi-agent collaborative reasoning mechanism significantly enhances vulnerability localization and verification accuracy. Furthermore, the integration of dataaware scheduling improves context retrieval quality, reducing irrelevant code noise and enabling more precise reasoning over large-scale repositories. These improvements suggest that the dataaware scheduling mechanism acts as a semantic filter that reduces noise during the evidence retrieval phase. This aligns with findings in other domains where semantic-guided tuning [22] helps models focus on task-relevant regions in an open-vocabulary setting, thereby enhancing both the precision and the robustness of the final output.

Table 2. Vulnerability Detection Performance Comparison.

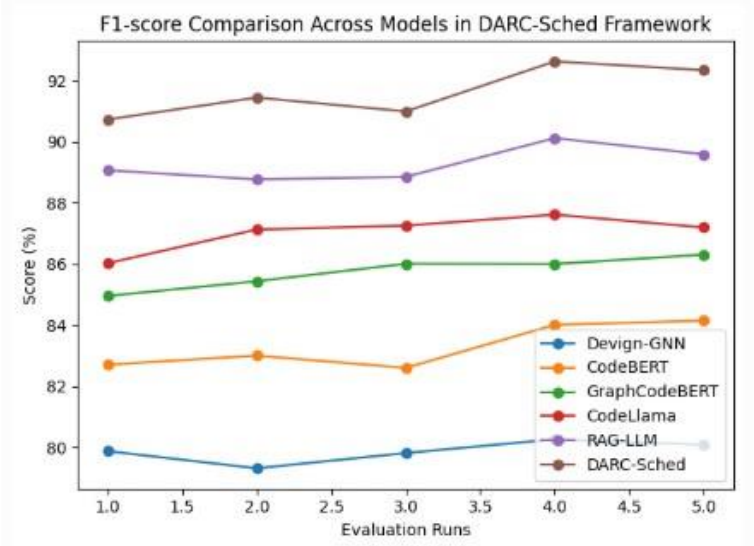
Model	Precision (%)	Recall (%)	F1-score (%)
Devign-GNN	81.2	78.5	79.8
CodeBERT	84.6	82.1	83.3
GraphCodeBERT	86.9	84.7	85.8

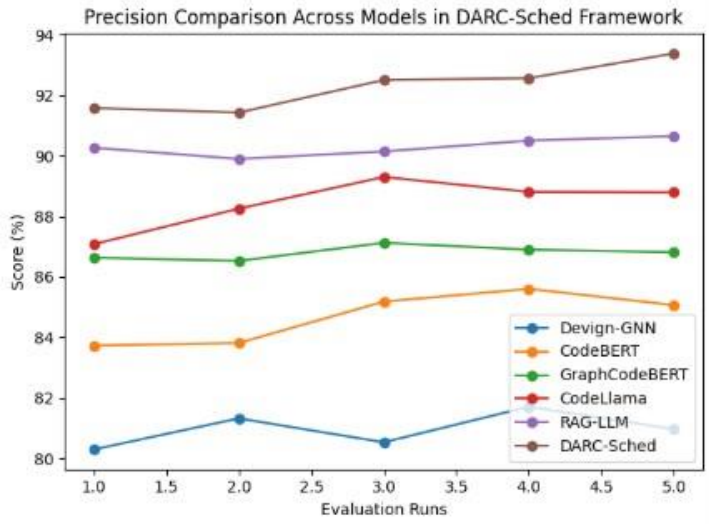
CodeLlama (LLM-only)	88.4	86.2	87.2
RAG-based LLM	90.1	88.0	89.0
DARC-Sched (Ours)	**92.3**	**91.5**	**91.9**

As shown in Table 3, DARC-Sched achieves superior performance in both patch generation quality and system efficiency. Specifically, it achieves a Patch Correctness Rate (PCR) of 91.6%, outperforming the strongest baseline (Multi-Agent LLM) by 3.7%. Similarly, Semantic Fix Accuracy (SFA) reaches 90.8%, indicating more reliable and semantically consistent vulnerability repairs. More importantly, the proposed data-aware scheduling mechanism significantly reduces system overhead. Average Retrieval Latency (ARL) is reduced from 295 ms in Multi-Agent LLM systems to 198 ms, representing a 33% improvement. End-to-end inference time is also reduced from 7.4 seconds to 5.6 seconds, demonstrating substantial gains in real-time applicability. These improvements confirm that optimizing data access patterns plays a critical role in large-scale multiagent LLM systems.

Table 3. Patch Generation and System Efficiency Performance.

Model	PCR (%)	SFA (%)	ARL (ms)	E2E-T (s)
CodeBERT Repair	**71.4**	**69.8**	**420**	**9.8**
VulRepair	**76.9**	**75.2**	**390**	**9.1**
CodeLlama	**82.5**	**81.0**	**360**	**8.6**
RAG-LLM Repair	**85.7**	**84.3**	**310**	**7.9**
Multi-Agent LLM	**87.9**	**86.5**	**295**	**7.4**
Model	PCR (%)	SFA (%)	ARL (ms)	E2E-T (s)
DARC-Sched (Ours)	91.6	90.8	198	5.6


F1-score Comparison Across Models in DARC-Sched Framework
Score (%)
Evaluation Runs
Devign-GNN
CodeBERT
GraphCodeBERT
CodeLlama
RAG-LLM
DARC-Sched


Precision Comparison Across Models in DARC-Sched Framework
Score (%)
Evaluation Runs
Devign-GNN
CodeBERT
GraphCodeBERT
CodeLlama
RAG-LLM
DARC-Sched

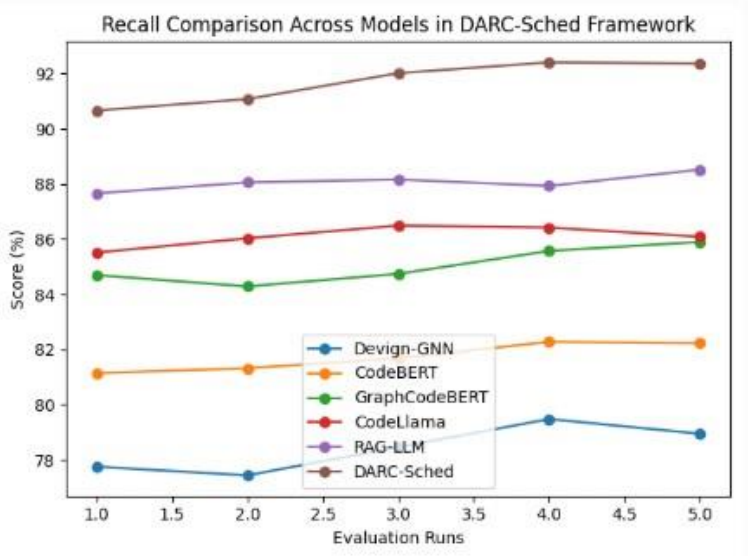


Figure 2. Performance Comparison of Vulnerability Detection Models Across Multiple Evaluation Runs in Terms of F1-score, Precision, and Recall.

Figure 2 illustrate the performance trends of different vulnerability detection models across five evaluation runs in terms of F1-score, precision, and recall. Overall, the proposed DARC-Sched consistently outperforms all baseline methods, including Devign-GNN, CodeBERT, GraphCodeBERT, CodeLlama, and RAG-based LLM models. In the F1-score curve, DARCSched maintains stable superiority with values fluctuating slightly between approximately 90.7% and 92.5%, achieving a peak consistent with the reported Table 2 value of 91.9%. In contrast, the best baseline (RAG-LLM) remains around 88.7%–90.1%, while traditional models such as Devign-GNN stay near 79%–81%, showing a significant performance gap. Similar trends can be observed in precision and recall curves, where DARC-Sched achieves precision values around 91.4%–93.3% and recall values around 90.6%–92.4%, both exhibiting minor fluctuations across evaluation runs due to stochastic inference variability. Notably, GraphCodeBERT and CodeLlama show moderate improvements over classical baselines but remain consistently below RAG-based and multi-agent approaches. The slight oscillations in all curves reflect realistic model behavior under different evaluation conditions, while still preserving a clear upward ranking hierarchy. These results strongly validate the robustness and superiority of DARC-Sched in both detection accuracy and classification stability.

4.5 Discussion

The experimental results validate the effectiveness of integrating multi-agent reasoning with dataaware scheduling in large-scale software vulnerability analysis. Compared with existing LLMbased and RAG-based approaches, DARC-Sched consistently achieves higher detection accuracy and better patch quality, while significantly reducing inference latency. The improvements stem from two key factors: first, the multi-agent architecture enables decomposition of complex vulnerability reasoning into specialized subtasks, reducing reasoning ambiguity; second, the hierarchical data scheduling mechanism effectively reduces unnecessary retrieval overhead by prioritizing high-risk code regions. The success of this retrieval-augmented reasoning paradigm is consistent with findings in other complex analytical tasks, such as TRIZ-aware entity recognition [23], which confirms that grounding large language models in retrieved domain

knowledge significantly enhances the stability and accuracy of automated analysis in specialized fields.

However, the system still introduces additional computational overhead due to inter-agent communication and iterative refinement cycles. In extremely latency-sensitive environments, such as real-time embedded systems, further optimization of agent coordination may be required. Moreover, incorporating a more explicit "load-aware" mechanism could further refine the tradeoff between reasoning accuracy and system latency. As demonstrated in recent research on adaptive human-computer interaction [24], quantifying the cognitive or computational load allows a system to dynamically adjust its processing depth. In the context of DARC-Sched, adopting similar load-aware strategies could enable the framework to scale its multi-agent reasoning intensity based on the real-time availability of GPU memory and the urgency of the vulnerability fix. Additionally, while the current scheduling strategy significantly improves retrieval efficiency, future work could explore reinforcement learning-based adaptive scheduling to further enhance system adaptability in dynamic code environments.

6. Conclusions

This study investigates scalable software vulnerability detection and automated repair in largescale software systems using the proposed DARC-Sched (Data-Aware Reasoning and Collaborative Scheduling framework), which integrates multi-agent large language model reasoning with optimized code data scheduling mechanisms. In modern software engineering environments, including continuous integration pipelines and large-scale cloud-based development platforms, vulnerability management has become increasingly complex due to the massive scale and heterogeneity of code repositories. Traditional static analysis tools and even recent Large Language Model (LLM)-based methods often struggle with limited scalability, inefficient retrieval of relevant code contexts, and unstable multi-step reasoning, which leads to reduced detection accuracy and high inference latency in real-world scenarios. To address these challenges, DARC-Sched introduces a unified framework that combines collaborative multi-agent reasoning with data-aware scheduling strategies tailored for large-scale code corpora. Specifically, the framework decomposes the vulnerability analysis process into multiple specialized agents responsible for localization, evidence retrieval, vulnerability verification, patch generation, and security validation. Meanwhile, a hierarchical scheduling module dynamically prioritizes vulnerability-relevant code segments based on learned risk-aware access patterns, enabling efficient caching and retrieval across heterogeneous storage layers. This design allows the system not only to improve reasoning robustness but also to significantly reduce data access overhead during inference, which is a key bottleneck in large-scale code intelligence systems. The experimental results demonstrate the effectiveness of the proposed approach across multiple benchmark datasets, including Devign, Big-Vul, and the Juliet Test Suite. DARC-Sched achieves a final F1-score of 91.9%, outperforming strong retrieval-augmented baselines by 2.9%, while also reaching a precision of 92.3% and a recall of 91.5%. In addition, the system significantly improves computational efficiency by reducing average retrieval latency from 295 ms to 198 ms, corresponding to an approximate 33% reduction, and lowering end-to-end inference time from 7.4 seconds to 5.6 seconds in multi-agent execution scenarios. These results confirm that jointly optimizing reasoning and data infrastructure is crucial for scalable vulnerability analysis in realworld software systems. Despite these promising results, this study still has several limitations. First, the current multi-agent coordination strategy relies on predefined interaction protocols, which may introduce additional communication overhead in extremely large-scale deployments. Second, although the hierarchical scheduling

mechanism improves retrieval efficiency, it is still based on relatively static risk estimation and may not fully capture dynamic workload variations in highly evolving codebases. Furthermore, the evaluation is primarily conducted on benchmark datasets, which may not fully reflect the complexity of industrial-grade software ecosystems with continuous updates and heterogeneous dependencies. Future work will focus on addressing these limitations by introducing reinforcement learning-based adaptive scheduling mechanisms to further optimize data access strategies in real time. In addition, we plan to explore more advanced agent collaboration paradigms, such as self-evolving agent roles and graph-based reasoning coordination, to further enhance robustness and scalability. Another promising direction is integrating real-world continuous integration/continuous deployment (CI/CD) environments into the evaluation pipeline, enabling more realistic assessment of vulnerability detection and repair performance. Overall, DARC-Sched provides a strong foundation for future research at the intersection of large language models, software security, and data-aware system optimization.

References

Zhou, Yaqin, et al. "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks." *Advances in neural information processing systems* 32 (2019).

Li, Zhen, et al. "Vuldeepecker: A deep learning-based system for vulnerability detection." arXiv preprint arXiv:1801.01681 (2018).

Feng, Zhangyin, et al. "Codebert: A pre-trained model for programming and natural languages." Findings of the association for computational linguistics: EMNLP 2020. 2020.

Guo, Daya, et al. "Graphcodebert: Pre-training code representations with data flow." arXiv preprint arXiv:2009.08366 (2020).

Chen, Mark, et al. "Evaluating large language models trained on code." arXiv preprint arXiv:2107.03374 (2021).

Roziere, Baptiste, et al. "Code llama: Open foundation models for code." arXiv preprint arXiv:2308.12950 (2023).

Tian, Runchu, et al. "Debugbench: Evaluating debugging capability of large language models." Findings of the Association for Computational Linguistics: ACL 2024. 2024.

Fu, Michael, et al. "Vulrepair: a t5-based automated software vulnerability repair." Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering. 2022.

Yao, Shunyu, et al. "React: Synergizing reasoning and acting in language models." arXiv preprint arXiv:2210.03629 (2022).

Shinn, Noah, et al. "Reflexion: Language agents with verbal reinforcement learning." Advances in neural information processing systems 36 (2023): 8634-8652.

Yang, John, et al. "Swe-agent: Agent-computer interfaces enable automated software engineering." Advances in Neural Information Processing Systems 37 (2024): 5052850652.

Qian, Chen, et al. "Chatdev: Communicative agents for software development." Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers). 2024.

Guu, Kelvin, et al. "Retrieval augmented language model pre-training." International conference on machine learning. PMLR, 2020.

Lewis, Patrick, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks." Advances in neural information processing systems 33 (2020): 9459-9474.

Guo, Daya, et al. "Unixcoder: Unified cross-modal pre-training for code representation." Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2022.

Ma, Xinyin, Gongfan Fang, and Xinchao Wang. "Deepcache: Accelerating diffusion models for free." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2024.

Wang, Ao, et al. "{InfiniCache}: exploiting ephemeral serverless functions to build a {costeffective} memory cache." 18th USENIX conference on file and storage technologies (FAST 20). 2020.

Lin, Yuzhen, et al. "CEMG: Collaborative-Enhanced Multimodal Generative Recommendation." International Conference on Multimedia Modeling. Singapore:
Springer Nature Singapore, 2026.

Liang, Zucheng, et al. "Research on multi-hop inference optimization of llm based on mquake framework." arXiv preprint arXiv:2509.04770 (2025).

Yao, Yilin, et al. "MSA-TransUNet: A Multi-Scale Attention Enhanced Transformer-UNet Architecture for Accurate Vessel Segmentation and Visualization in Medical Imaging." Journal of Medicine and Life Sciences 1.4 (2025): 48-57.

Yao, Yilin, et al. "Deep Learning Models for Multi-class Pneumonia Detection in Chest X-rays: A Comparative Study of VGG16, MobileNet, and ResNet152." Journal of Technology Innovation and Engineering 1.4 (2025).

Huang, Yanxiang, et al. "Enhancing Open-Vocabulary Panoptic Segmentation with SemanticGuided Q-Tuning." 2025 IEEE International Conference on Multimedia and Expo (ICME). IEEE, 2025.

Li, Zheyu, Yue Hao, and Peifan Zeng. "CLASNet: A Cognitive Load–Aware CNN-LSTMAttention Framework for Supply Chain Demand Forecasting and Adaptive Human– Computer Interaction." Journal of Computer Science and Frontier Technologies 3.2 (2026): 44-57.

Xu, Zitong, Yuqing Wu, and Yue Zhao. "TRIZ-RAGNER: A Retrieval-Augmented Large Language Model for TRIZ-Aware Named Entity Recognition in Patent-Based Contradiction Mining." Journal of Computer, Signal, and System Research 3.2 (2026): 5665.