Towards Budget-Aware Early Candidate Selection for Vulnerability Analysis

Xinye Zhao
University of Notre Dame
Notre Dame, USA
xzhao24@nd.edu

Joanna C. S. Santos
University of Notre Dame
Notre Dame, USA
joannacss@nd.edu

Abstract

Actionable warning identification (AWI) prioritizes static-analysis warnings under limited review budgets, but most work ranks final warnings or complete source-to-sink paths. These artifacts are expensive to produce and are typically unavailable early in a vulnerability-analysis pipeline. We study budget-aware early candidate selection from lightweight candidates, including suspicious sinks, source–sink fragments, and vulnerability-looking methods. We define budget as the percentage of candidates that downstream validation can inspect and compare heuristic, learned, and diversity-aware ranking policies. On CWE-Bench-Java, diversity-aware random-forest ranking recovers 62.5% of oracle-covered vulnerable samples at a 10% candidate budget, compared with 29.1% for random selection. On OWASP Benchmark Java and Semgrep-derived OWASP candidates, ranking also improves low-budget sample discovery under case-level proxy labels. These results suggest that early candidate ranking can better allocate limited validation effort, while future work should connect this stage to concrete downstream validation costs.

CCS Concepts

• **Security and privacy** → **Software security engineering**; • **Software and its engineering** → *Software verification and validation*; • **Computing methodologies** → Machine learning.

Keywords

vulnerability analysis, static analysis, candidate prioritization, budget-aware ranking, software security

ACM Reference Format:
Xinye Zhao and Joanna C. S. Santos. 2026. Towards Budget-Aware Early Candidate Selection for Vulnerability Analysis. In *Proceedings of the 2nd Workshop on Explainable and Reliable Software Systems (EXPRESS '26), October 4–9, 2026, Oakland, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3842650.3843175

1 Introduction

Vulnerability analysis is crucial for software security, but practical vulnerability analysis pipelines face limited validation budgets, including analyst review time, path-construction effort, symbolic-analysis resources, and large language model (LLM) query costs. Static analyzers can cheaply produce many warnings or suspicious code locations, while downstream validation is much more expensive [8–11]. Therefore, the order in which evidence is sent downstream can affect which vulnerabilities are ultimately discovered.

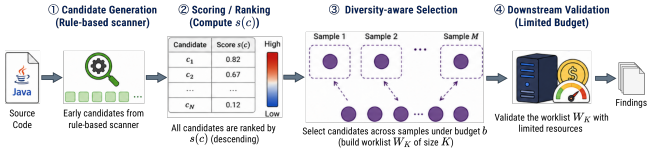

Figure 1: Overview of our budget-aware early candidate selection approach.

Actionable warning identification (AWI) addresses this problem by ranking or filtering warnings so analysts focus on the most likely relevant alerts [5, 6]. However, most AWI work operates on *final* warnings, after the analyzer has already constructed a report. Related triage work also often assumes complete warnings, slices, or source-to-sink flows are already available [12, 14, 15]. We study an earlier decision point in a vulnerability analysis pipeline: before a complete vulnerability path or final warning is produced, the analysis may already observe many lightweight candidates. These include suspicious security-sensitive operations, or *sinks*, such as file access, SQL queries, or command execution; attacker-controlled inputs, or *sources*, such as request parameters, headers, or file names; partial source–sink connections; or methods that contain vulnerability-like code patterns.

We formulate this problem as *budget-aware early candidate selection*. Given a large candidate pool and a limited downstream validation budget, the goal is to select a small worklist that preserves as many discoverable vulnerabilities as possible. This differs from false-positive removal because selected candidates are not final bug reports; they are inputs scheduled for deeper analysis. By ranking candidates with visible signals, such as dangerous API use, attacker-controlled input, local context, tool agreement, and diversity across samples, the selected worklist also gives analysts a more transparent view of why each candidate is sent to downstream validation under a limited review budget. [3, 18, 19].

We study transparent heuristic ranking policies and lightweight learning-based rankers, with and without diversity-aware selection. In a preliminary study on CWE-Bench-Java [11], a random-forest ranker that selects at most one high-scoring candidate per sample recovers 62.5% of oracle-reachable vulnerable samples at a 10% candidate budget, more than doubling random selection. We also use OWASP Benchmark Java [16] as a supplementary case-level sanity check.

Our contributions are as follows:

EXPRESS '26, Oakland, CA, USA

ACM ISBN 979-8-4007-2968-3/2026/10
https://doi.org/10.1145/3842650.3843175

- We formulate early vulnerability-candidate selection as a budgeted worklist problem before final warnings or complete source-to-sink paths are available.
- We compare transparent heuristic rankers, lightweight learned rankers, and diversity-aware variants under the same candidate budgets.
- We evaluate on CWE-Bench-Java and OWASP Benchmark Java, distinguishing method-level labels from case-level proxy labels.

2 Methodology

2.1 Problem Formulation

Given a dataset of program samples $P = \{p_1, \ldots, p_m\}$, a lightweight analyzer produces a set of early vulnerability candidates for each sample, $C_p = \{c_{p,1}, \ldots, c_{p,n_p}\}$. Let $C = \bigcup_{p \in P} C_p$ be the global candidate pool. In this paper, the downstream budget is defined as the limited percentage of candidates that a validator can afford to inspect. For a budget fraction b, this corresponds to a worklist size $K = \lfloor b|C| \rfloor$. This candidate-percentage budget is a controlled proxy for real-world validation resources, such as analyst time, compute cost, LLM-query cost, or monetary cost. The ranking task is therefore to select a worklist $W_K \subseteq C$ with $|W_K| \leq K$.

This selection balances candidate quality with sample-level discovery: the worklist should prioritize likely useful candidates while reaching as many distinct vulnerable samples as possible under the same budget.

We write $hit(c) = 1$ if candidate c reaches the available ground-truth signal. We evaluate candidate quality with $Precision@K$ and sample-level discovery with normalized coverage, defined in the evaluation. Thus, the worklist objective is to balance candidate quality with vulnerable-sample coverage under a fixed downstream validation budget.

2.2 Budgeted Early Candidate Selection

Each candidate c has a score $s(c)$ estimating its usefulness for downstream validation. We compare four lightweight ***scoring schemes***: *Random* sampling, *SinkOnly* by sink severity, *SourceSink* by attacker-controlled source evidence, and *FullContext* using candidate type, tool support, path length, security keywords, and sanitizer-like terms. Moreover, we use logistic regression, linear SVM, random forest, and ExtraTrees because linear, margin-based, and tree-ensemble models have shown strong performance in AWI and effort-aware defect-ranking tasks [5, 7, 21, 23]. These models use only early candidate features, including candidate type, vulnerability family, sink category, source evidence, sanitizer indicators, path length, tool support, statement length, and heuristic scores. They define $s(c)$ for ranking, not final vulnerability classification.

To avoid spending the budget on many candidates from the same sample, we apply ***diversity-aware selection***. For each sample p, we keep its highest-scoring candidate, $c_p^* = \arg\max_{c \in C_p} s(c)$, rank samples by $s(c_p^*)$, and form W_K from the top-ranked samples. Thus, scoring estimates candidate quality, while diversity allocates the validation budget across samples.

3 Preliminary Evaluation

3.1 Experimental Setup

Datasets. We evaluate on two Java vulnerability benchmarks. Our main benchmark is CWE-bench-java [11] where we use 48 vulnerability instances to evaluate method-level candidate localization. As a supplementary benchmark, we use 300 OWASP Benchmark Java test cases to evaluate case-level ranking behavior. The evaluated CWE scope is summarized in Appendix B. OWASP does not provide reliable line-level ground truth, so we use it only as a proxy sanity check rather than as the main localization benchmark.

Candidate sources. Our ranking method takes as input a pool of early vulnerability candidates. In **CWE-Bench-Java, Ours** and **OWASP, Ours**, this pool is produced by our recall-oriented rule-based scanner; in **OWASP, Semgrep**, it is produced by Semgrep and then ranked using the same policies. This design separates candidate generation from candidate selection: the generator determines which vulnerabilities are reachable by any ranker, while the ranker determines how effectively a limited budget is allocated within that reachable pool. Accordingly, we report coverage with respect to the generated-candidate oracle rather than all benchmark vulnerabilities.

Ranking policies. We compare ***Random***, a baseline with no prioritization averaged over 30 trials, with three evidence-based rankers. ***SinkOnly*** uses sink severity, ***SourceSink*** adds attacker-controlled source evidence and local source–sink relations, and ***FullContext*** adds candidate type, tool support, path length, same-file relation, security keywords, and sanitizer-like terms. These named features give analysts compact context for understanding a candidate beyond its detected source-code location. For any scoring policy, ***+Div.*** denotes the diversity-aware variant, which first keeps the highest-scoring candidate per sample and then ranks samples by that candidate score.

Learning-based rankers. We evaluate several lightweight supervised models, including logistic regression, linear SVM, random forest, and ExtraTrees, and report the strongest representative result in the main table. The models use candidate-level features available before downstream validation, including candidate type, vulnerability family, sink category, source evidence, sanitizer indicators, path length, tool support, source/sink statement length, and heuristic scores. To avoid leakage across candidates from the same vulnerability instance, we use project-level grouped cross-validation and rank candidates using out-of-fold prediction scores. We evaluate learning-based rankers only on CWE-Bench-Java, where method-level labels support candidate-level supervision; OWASP lacks reliable line-level ground truth, so we use it only for heuristic proxy evaluation.

Metrics. We report results under 10% and 20% global candidate budgets, with $K = \lfloor b|C| \rfloor$ for budget fraction b. These low-budget settings are where prioritization is most useful. Effort-aware defect prediction and fine-grained localization studies commonly evaluate ranking methods by measuring how many defects are recovered after inspecting only a limited fraction of code, changed lines, or review effort [1, 7, 17, 20]. We use 10% as a stricter early-budget setting and 20% as a common effort-aware reference point.

Table 1: Low-budget headline results. Coverage is normalized sample-level coverage; Precision is candidate-level precision.

CWE-Bench-Java, Ours				OWASP, Ours				OWASP, Semgrep			
Ranker	Budget	Coverage	Precision	Ranker	Budget	Coverage	Precision	Ranker	Budget	Coverage	Precision
Random	10%	0.291	0.299	Random	10%	0.159	0.066	Random	10%	0.207	0.427
SourceSink+Div.	10%	0.500	**0.469**	SinkOnly+Div.	10%	**0.694**	**0.277**	SinkOnly+Div.	10%	0.414	0.730
FullContext+Div.	10%	0.406	0.367	SourceSink+Div.	10%	0.682	0.272	SourceSink+Div.	10%	0.414	0.730
ML-RF+Div.	10%	**0.625**	0.408	FullContext+Div.	10%	0.659	0.263	FullContext+Div.	10%	**0.423**	**0.746**
Random	20%	0.472	0.312	Random	20%	0.295	0.063	Random	20%	0.365	0.419
SourceSink+Div.	20%	0.531	0.439	SinkOnly+Div.	20%	0.871	0.192	SinkOnly+Div.	20%	0.505	0.441
FullContext+Div.	20%	0.500	0.388	SourceSink+Div.	20%	0.906	0.208	SourceSink+Div.	20%	0.505	0.441
ML-RF+Div.	20%	**0.781**	**0.449**	FullContext+Div.	20%	**0.965**	**0.227**	FullContext+Div.	20%	**0.613**	**0.535**

For a selected worklist W_K of size K, candidate-level precision is

$$\text{Precision@K} = \frac{|\{c \in W_K \mid \text{hit}(c) = 1\}|}{|W_K|}.$$

We measure sample-level discovery by normalized coverage:

$$\text{NormCov@K} = \frac{|\{p \mid \exists c \in W_K \cap C_p,\ \text{hit}(c) = 1\}|}{|\{p \mid \exists c \in C_p,\ \text{hit}(c) = 1\}|}.$$

The denominator is the oracle coverage of the generated candidate pool: the number of samples that could be reached if all generated candidates were selected. On CWE-Bench-Java, $\text{hit}(c)$ means overlap with a fixed vulnerable method region, which is a method-level localization proxy rather than proof of the exact vulnerable statement or complete source-to-sink flow. On OWASP, $\text{hit}(c)$ is a case-level proxy based on the expected vulnerability family, so OWASP results should be interpreted as supplementary ranking evidence rather than precise vulnerability localization.

3.2 Research Questions

Our evaluation answers three research questions (RQs):

- **RQ1** ***Can budget-aware ranking improve vulnerable-sample discovery over random selection?***
- **RQ2** ***Does diversity-aware selection improve sample-level discovery under a global candidate budget?***
- **RQ3** ***Does the ranking behavior generalize beyond our own candidate generator to an external SAST candidate pool?***

4 Results

4.1 RQ1: Budgeted Discovery

Table 1 shows that the selected worklist is more useful when candidates are prioritized rather than sampled uniformly. On CWE-Bench-Java, ML-RF+Div., a random-forest ranker with diversity-aware selection, recovers a larger fraction of the candidate pool's oracle-reachable samples than random selection at both 10% and 20% budgets. This indicates that useful candidates are not uniformly distributed in the pool; early evidence such as source, sink, context, and learned candidate features can identify candidates that are more likely to preserve downstream discoveries. The OWASP proxy results show the same pattern with a different label granularity: ranking concentrates candidates that match the expected vulnerability category near the top of the worklist. Overall, these results show that early candidate selection can improve budget allocation before downstream validation.

RQ1 Finding: **Budget-aware ranking preserves more vulnerable samples than random selection under small candidate budgets.**

Table 2: CWE-Bench-Java diversity ablation at 10% budget.

Ranker	Unique	Hit	Recall
SinkOnly	9	6	0.188
SinkOnly+Div.	40	18	**0.562**
SourceSink	15	8	0.250
SourceSink+Div.	40	16	**0.500**
FullContext	18	7	0.219
FullContext+Div.	40	12	**0.375**

4.2 RQ2: Effect of Diversity

Table 2 isolates the effect of diversity under the same 10% candidate budget. Score-only ranking can concentrate many selected candidates in a small number of samples; for example, SinkOnly reaches only 9 unique samples, while SinkOnly+Div. reaches 40. This broader allocation improves sample-level recall for all three rankers, increasing SinkOnly from 0.188 to 0.562, SourceSink from 0.250 to 0.500, and FullContext from 0.219 to 0.375. These results show that diversity improves the use of a global validation budget by reducing repeated selections from the same sample.

RQ2 Finding: **Diversity-aware selection converts candidate-level ranking into sample-level discovery.**

4.3 RQ3: Generality

To test whether our results depend on artifacts of our own candidate generator, we evaluate the same ranking policies on Semgrep candidates from OWASP. FullContext+Div. improves both low-budget coverage and precision over random selection, suggesting that the benefit comes from the budget-aware selection objective rather than only from our scanner's output order or rule design. However, OWASP provides case-level labels rather than line-level ground truth, so this experiment is supplementary; CWE-Bench-Java remains our main localization benchmark.

RQ3 Finding: **The ranking behavior also appears on an external SAST candidate pool.**

5 Discussion

Early ranking is feasible. Our results suggest that early candidate ranking is a useful decision point in vulnerability-analysis pipelines. Even with lightweight rule-based candidate generation and simple ranking policies, the selected worklists preserve more vulnerable samples than random selection under small budgets. This means that expensive downstream validation does not need to start from an unstructured candidate pool. Instead, a cheap front-end can already provide useful signals for where deeper analysis should

focus, before a complete warning, exploit trace, or source-to-sink path has been constructed.

Diversity matters. Suspiciousness alone is not enough. A candidate may look highly risky, but selecting many candidates from the same sample can waste a global validation budget and leave other vulnerable samples unreached. Diversity-aware ranking is one simple way to better match the downstream objective of reaching distinct vulnerable samples under the same candidate budget. Other allocation strategies may account for candidate difficulty, sample size, or expected validation cost, but the results show that early candidate selection should reason beyond candidate-level suspiciousness alone.

Ranking evidence is inspectable. The heuristic rankers expose the evidence used for prioritization, such as source evidence, sink severity, sanitizer hints, path/locality, tool support, and security-related keywords. These signals do not prove exploitability, but they give analysts compact context for why a candidate was placed in the worklist. This makes the ordering more inspectable than an opaque tool order and provides a basis for future analyst-facing studies of explanation usefulness and validation effort.

6 Related Work

Actionable warning identification (AWI). Actionable warning identification ranks or filters static-analysis warnings so limited review effort is spent on alerts that are more likely to be useful to developers or security analysts [5, 6, 9]. Our work shares AWI's budgeted-review motivation, but targets an earlier stage of the analysis pipeline. Instead of assuming that final warnings or complete source-to-sink paths have already been produced, we ask which lightweight candidates should be selected before expensive downstream validation begins.

Budgeted and diversified ranking. Prior work on diversified ranking, maximum-coverage selection, active search, warning prioritization, and effort-aware learning-to-rank studies how to allocate limited inspection effort across ranked items [2, 4, 7]. These areas study how to allocate limited inspection effort across a ranked pool, especially when repeatedly selecting similar items can reduce the value of the worklist. We adapt this idea to vulnerability candidate selection by combining candidate-level evidence with sample-level diversity, so the selected budget is not consumed by many candidates from the same program sample.

Candidate and slice-based vulnerability detection. Prior vulnerability detection systems often represent code through candidate statements, slices, paths, or other intermediate units before classification. For example, VulDeePecker [13] and SySeVR [12] extract vulnerability-related code slices for learning-based detection. Recent LLM-based vulnerability detectors also use code snippets, intermediate representations, or contextual evidence to improve vulnerability classification [14, 18, 22]. We build on the observation that vulnerability evidence can be represented as candidates, but study a different objective: ranking early candidates to allocate limited downstream analysis budget.

Taint-flow and path triage. The closest recent work is by Ni *et al.* [15], which triages already constructed dynamic taint flows, i.e., relatively complete source-to-sink paths. Such flows are highly informative, but constructing them can itself require substantial analysis effort and may be unavailable early in the pipeline. Our work instead studies the earlier scheduling question: given partial sources, sinks, local context, and incomplete source–sink evidence, which candidates should be sent first to deeper validation?

7 Threats to Validity

Our evaluation focuses on the precision and coverage of selected candidates under a global candidate budget. This abstracts away differences in downstream analysis cost across vulnerability types. In practice, validation cost may vary across vulnerability types and candidate contexts: some candidates require long source-to-sink reasoning or multi-method analysis, while others can be checked from a local sink pattern. Therefore, a fixed top-K candidate budget may not fully capture the true cost of downstream validation.

Our labels are also proxies for downstream discovery. CWE-Bench-Java provides method-level vulnerable regions, so a candidate hit indicates overlap with the available vulnerable region rather than proof of the exact vulnerable statement or complete exploit flow. OWASP provides case-level expected vulnerability families, which are useful for a supplementary ranking sanity check but should not be interpreted as precise vulnerability localization.

Finally, our one-candidate-per-sample diversity policy is a simple allocation heuristic. Real deployments may need to vary budget by candidate difficulty, sample size, code complexity, or vulnerability family. We focus here on lightweight heuristic and supervised ranking policies; future work should compare them with selective cheap-LLM rankers that add semantic judgment only when the expected ranking benefit justifies the additional query cost.

8 Conclusion and Future Work

We formulated early vulnerability-candidate selection as a budgeted worklist problem before expensive downstream validation. Our preliminary results show that diversity-aware ranking preserves more vulnerable samples than random selection under small candidate budgets, while keeping rankings inspectable through source, sink, context, and diversity evidence. These results suggest that useful prioritization signals are already available before complete warnings or source-to-sink paths are constructed. Future work will connect this selection stage to concrete downstream validators, measure end-to-end costs such as analyst time, path-construction effort, and LLM-query cost, model CWE-specific validation costs, and evaluate the approach on larger multi-vulnerability projects.

A Early Candidate Extraction

Our candidate generator is a lightweight, recall-oriented scanner rather than a full vulnerability validator. For each Java sample, it extracts source-like evidence from externally influenced inputs, such as web request accessors, file and path names, environment or property lookups, command or script accessors, URL-related values, streams, deserialization inputs, and suspicious public method parameters. It then identifies sink-like statements associated with common Java vulnerability families, including path traversal, command injection, SQL injection, XSS and web output, code injection, deserialization, XXE, SSRF, open redirect, weak cryptography or randomness, TLS validation, and information exposure. The scanner emits short source-to-sink candidates when local propagation

Table 3: Benchmark vulnerability-type scope.

CWE	Vulnerability type	CWE-Bench	OWASP
CWE-022	Path traversal	27	32
CWE-078	OS command injection	2	22
CWE-079	Cross-site scripting	8	40
CWE-089	SQL injection	1	47
CWE-090	LDAP injection	–	5
CWE-094	Code injection	6	–
CWE-287	Improper authentication	1	–
CWE-327	Weak cryptography	–	38
CWE-328	Weak hashing	–	32
CWE-330	Weak randomness	–	61
CWE-501	Trust-boundary violation	–	10
CWE-502	Deserialization	1	–
CWE-614	Missing secure cookie flag	–	9
CWE-643	XPath injection	–	4
CWE-918	Server-side request forgery	2	–
Total	–	**48**	**300**

or nearby context connects a source to a sink, and sink-only recall candidates when only a high-value sink is observed. Each candidate is normalized with source and sink locations, statements, family, type, path length, sanitizer hints, and tool-support metadata. Near-duplicates with the same family and sink location are fused before ranking.

B Evaluation Test Cases

Table 3 summarizes the CWE scope of the evaluated datasets. Because the distributions are not uniform, we interpret our results as ranking performance over these candidate pools rather than as per-CWE performance claims.

References

[1] Peter Bludau and Alexander Pretschner. 2022. Feature sets in just-in-time defect prediction: an empirical evaluation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*. 22–31.

[2] Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. 2008. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 659–666.

[3] Xueying Du, Geng Zheng, Kaixin Wang, Yi Zou, Yujia Wang, Wentai Deng, Jiayi Feng, Mingwei Liu, Bihuan Chen, Xin Peng, et al. 2024. Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag. *ACM Transactions on Software Engineering and Methodology* (2024).

[4] Roman Garnett, Yamuna Krishnamurthy, Xuehan Xiong, Jeff Schneider, and Richard Mann. 2012. Bayesian optimal active search and surveying. *arXiv preprint arXiv:1206.6406* (2012).

[5] Xiuting Ge, Chunrong Fang, Xuanye Li, Weisong Sun, Daoyuan Wu, Juan Zhai, Shang-wei Lin, Zhihong Zhao, Yang Liu, and Zhenyu Chen. 2024. Machine learning for actionable warning identification: A comprehensive survey. *Comput. Surveys* 57, 2 (2024), 1–35.

[6] Xiuting Ge, Chunrong Fang, Quanjun Zhang, Daoyuan Wu, Bowen Yu, Qirui Zheng, An Guo, Shangwei Lin, Zhihong Zhao, Yang Liu, et al. 2025. Pre-trained model-based actionable warning identification: A feasibility study. *ACM Transactions on Software Engineering and Methodology* (2025).

[7] Yuchen Guo, Martin Shepperd, and Ning Li. 2024. Improving classifier-based effort-aware software defect prediction by reducing ranking errors. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. 160–169.

[8] Mete Keltek, Rong Hu, Mohammadreza Fani Sani, and Ziyue Li. 2024. Boosting cybersecurity vulnerability scanning based on llm-supported static application security testing. *arXiv preprint arXiv:2409.15735* (2024).

[9] Sunghun Kim and Michael D Ernst. 2007. Which warnings should I fix first?. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 45–54.

[10] Fengjie Li, Jiajun Jiang, Dongchi Chen, and Yingfei Xiong. 2026. LLM-based Vulnerability Detection at Project Scale: An Empirical Study. *arXiv preprint arXiv:2601.19239* (2026).

[11] Ziyang Li, Saikat Dutta, and Mayur Naik. 2025. IRIS: LLM-assisted static analysis for detecting security vulnerabilities. In *International Conference on Learning Representations*, Vol. 2025. 35735–35758.

[12] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. 2021. Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing* 19, 4 (2021), 2244–2258.

[13] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. Vuldeepecker: A deep learning-based system for vulnerability detection. *arXiv preprint arXiv:1801.01681* (2018).

[14] Andrew A Mahyari. 2024. Harnessing the power of llms in source code vulnerability detection. In *MILCOM 2024-2024 IEEE Military Communications Conference (MILCOM)*. IEEE, 251–256.

[15] Ronghao Ni, Aidan Z. H. Yang, Min-Chien Hsu, Nuno Sabino, Limin Jia, Ruben Martins, Darion Cassel, and Kevin Cheang. 2025. Learning to Triage Taint Flows Reported by Dynamic Program Analysis in Node.js Packages. *ArXiv* abs/2510.20739 (2025). https://api.semanticscholar.org/CorpusID:282304743

[16] OWASP Foundation. [n. d.]. OWASP Benchmark Project. https://owasp.org/www-project-benchmark/. Accessed: 2026-06-17.

[17] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. 2021. Jitline: A simpler, better, faster, finer-grained just-in-time defect prediction. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 369–379.

[18] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Wei Ma, Lyuye Zhang, Yang Liu, and Yingjiu Li. 2024. Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning. *arXiv preprint arXiv:2401.16185* (2024).

[19] Karl Tamberg and Hayretdin Bahsi. 2025. Harnessing large language models for software vulnerability detection: A comprehensive benchmarking study. *IEEE Access* (2025).

[20] Supatsara Wattanakriengkrai, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Hideaki Hata, and Kenichi Matsumoto. 2020. Predicting defective lines using a model-agnostic technique. *IEEE Transactions on Software Engineering* 48, 5 (2020), 1480–1496.

[21] Xueqi Yang, Jianfeng Chen, Rahul Yedida, Zhe Yu, and Tim Menzies. 2021. Learning to recognize actionable static code warnings (is intrinsically easy). *Empirical Software Engineering* 26, 3 (2021), 56.

[22] Yanjing Yang, Xin Zhou, Runfeng Mao, Jinwei Xu, Lanxin Yang, Yu Zhangm, Haifeng Shen, and He Zhang. 2024. DLAP: A Deep Learning Augmented Large Language Model Prompting Framework for Software Vulnerability Detection. *J. Syst. Softw.* 219 (2024), 112234. https://api.semanticscholar.org/CorpusID:269502008

[23] Rahul Yedida, Hong Jin Kang, Huy Tu, Xueqi Yang, David Lo, and Tim Menzies. 2023. How to find actionable static analysis warnings: A case study with findbugs. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2856–2872.

Received 2026-06-28; accepted 2026-07-31