LOOK HERE, THEN DECIDE: SMALL DECISION MODELS IN A HARNESS FIND MEMORY-SAFETY VULNERABILITIES AT SCALE ON COMMODITY GPUS

Tim De Wachter
Cyrex; Deus Rex
tim@deusrex.ai

Kyryl Volkov
Cyrex; Deus Rex
kyryl.volkov@deusrex.ai

Myrto Eleftheria Gkogkou
Deus Rex
myrto.gkogkou@deusrex.ai

Mathieu Huysman
Cyrex; Deus Rex
mathieu@deusrex.ai

ABSTRACT

We ask whether small, open-weight *decision models* can support vulnerability discovery in open-source software at scale and what the surrounding system must provide. These models answer a typed question in one forward pass, returning scores for the permitted answers rather than text. Our harness selects widely packaged C and C++ projects and uses deterministic incomplete-fix miners and shape localizers to decide *where to look*. Intern-Decision-4B, a 4B-parameter open-weight decision model, then scores every candidate function. The scores *rank* the queues rather than accept or reject candidates, apart from a low reach-score floor that some queues kept. The search uses no frontier model.

Alone, the model is weak: it separates vulnerable from benign functions with AUC 0.73, but a vulnerable function from its own patch with only 0.56. The model is more useful for ordering candidates. On 75 public CVEs from projects without any of our findings, the fixed function reached its repository's top 25 in 21% of cases—1.4 times as often as under random order of the same queues. Recall is bounded mainly by where the localizers look. On the campaign's own queues the model's order placed 7.5 times more confirmed findings in the top 10 than random order, an observed, selection-biased upper bound, because those queues were chosen for containing a confirmed finding and were reviewed in the model's order. On three desktop-class GB10 systems, the harness searched more than 4,000 projects, mining them at 900–1,100 repositories per hour across two of the systems. The campaign confirmed 187 memory-safety findings; by worst-case CVSS v3.1, 12 are critical, 148 high, 23 medium and 4 low. The decision step runs at 4.4 decisions per second on one GB10 and at 9.8 on a consumer RTX 4090 gaming PC, with matching scores. For this model and harness, a small decision model supports vulnerability discovery through prioritization, while deterministic analysis and verification remain essential. All confirmed findings are being disclosed to maintainers under coordinated disclosure before publication.

Keywords Memory safety · Vulnerability discovery · Decision models · Static analysis · Candidate prioritization · Open-source software

1 Introduction

C and C++ underpin operating systems, libraries and network services used by millions of people. Much of this code handles untrusted files or network data, and a mistake in a memory operation can become a security vulnerability. Memory corruption has long persisted in C and C++ software [1], and memory-safety defects remain the largest class of serious vulnerabilities in such code [2, 3]. A defect in a widely packaged library can affect many downstream applications. Finding these defects early gives maintainers time to coordinate fixes.

Static application security testing (SAST) is one of the most widely used approaches. Rule engines such as Semgrep, its open fork Opengrep, and CodeQL [4, 5, 6, 7] inspect source code without running it. They use syntax trees and data-flow analysis to find code that matches a known-bad pattern. These tools are fast and explainable, but do not

decide whether the flagged code is reachable or exploitable. Empirical studies also report high noise and large recall gaps on real C vulnerabilities [8, 9].

Large language models (LLMs) offer another route to vulnerability discovery. Recent agentic systems combine language models with execution environments and analysis tools to investigate and reproduce suspected vulnerabilities [10, 11]. Anthropic later introduced Claude Mythos Preview, a frontier model available to selected partners, and reported its ability to identify high-severity vulnerabilities in widely used software [12, 13]. OpenAI reported that a single run of a cyber-specialised frontier model validated and wrote patches for 948 local privilege-escalation vulnerabilities in the Linux kernel, at a cost it gives as US$1,213,430 in API credits, about US$1,280 per vulnerability [14, 15]. These developments show what LLM-based discovery can do. Applying frontier models to thousands of repositories, however, is expensive when each investigation needs many model calls and tool executions; access to some of the most capable models also remains restricted. Large-scale discovery is therefore hard for teams with limited resources.

1.1 Motivation

How can language models be used across large codebases without an expensive, in-depth analysis of every candidate? Running an agent on every function of thousands of repositories would need substantial compute, and most of the functions it read would contain no vulnerability. Deterministic analysis can narrow the search before a small model decides which candidates to investigate first.

In 2026 a model category suited to the second step appeared: the *decision model*, or *System-One* model, a category TypeSafe AI introduced with its Jev model [16]. Instead of producing an explanation, a decision model evaluates predefined questions and returns a score for each permitted answer in a single forward pass (Section 2). Each decision costs one prefill and no decoding. Its output orders a queue directly, with nothing to parse.

We study Intern-Decision-4B, a small open-weight decision model [17], as one component of a larger harness rather than as a standalone detector. Our central research question is:

> *Can small, open-weight decision models improve the prioritization of potential memory-safety vulnerabilities at scale, even when their standalone classification performance is limited? And what must the harness around them do?*

1.2 Threat model

We take the defender's view. Consider a team with a few desktop-class GPU systems, open-weight models and commodity build infrastructure, but no access to frontier models restricted to selected partners. Can it find memory-safety defects in widely packaged C and C++ software early, at a rate that helps maintainers fix them first? Our results suggest it can, for this model and harness. We read them as showing that the design of the harness matters more than the size of the model; that is an interpretation, not a controlled comparison of model sizes. The ranking ablation and recall study (Sections 6.2 and 6.3) measure parts of this interpretation; the evaluation does not isolate every component. All confirmed findings are being disclosed to maintainers under coordinated disclosure before publication (Section 7).

1.3 Contributions

Our harness combines deterministic candidate generation, model-based ranking and subsequent verification. It selects projects by deployment reach and existing fuzzing coverage, mines their revision histories for incomplete security fixes, and searches for code shapes associated with unsafe memory operations. For example, a security patch may add a bounds check before a copy in one function while leaving a similar copy elsewhere unchanged; that sibling is a candidate, though not necessarily a vulnerability. Intern-Decision-4B scores every candidate. Stronger coding agents and human reviewers then verify the candidates they read. Candidate generation decides where to look, the decision model decides what to read first, and verification decides whether a vulnerability exists. Our contributions are:

1. **An evaluation of small decision models as vulnerability classifiers.** On real CVE fixes, Intern-Decision-4B and six configuration baselines built from other small open classifiers separate vulnerable code from benign code but barely from its own patch (Section 6.1).
2. **The harness that makes a weak model useful.** Deterministic localizers and miners, a reach-ranked and under-fuzzed corpus, and ranking in place of thresholds. In localizer pilots, 30–33% of the top 30 of a ranked queue were real candidates, against 6.7% of a random sample; this is a yield observation by configuration, not an isolated ranking effect (Sections 4 and 6.2).
3. **Measured ranking lift and recall.** We measure recall on 75 randomly drawn public CVEs from projects without any of our findings: ranking placed fixed functions within the review budget 1.3–1.4 times as often

as random order of the same queues. An ablation on the campaign's real queues found 7.5 times as many confirmed findings in the top 10 as random order, an observed, selection-biased upper bound; we also publish the CVE selection rule and list (Sections 6.2 and 6.3).
4. **Speed on commodity hardware.** 4.4 decisions per second on one GB10 and 9.8 on a consumer RTX 4090 gaming PC under vLLM, with matching scores on both and on a laptop GPU, and a cost-and-shape comparison with Opengrep (Section 6.4).
5. **A large-scale campaign.** More than 4,000 open-source C and C++ projects searched on three desk-side GPU systems, with mining at 900–1,100 repositories per hour: 187 confirmed memory-safety findings, with their worst-case CVSS v3.1 severity and every false-positive measure the record supports (Section 6.5).

2 Background

2.1 Decision models and System-One-style inference

We use *decision model* for a language model used as a classifier: it answers a predefined question in one forward pass, without generating text. The term and the model category were introduced by TypeSafe AI with Jev [16]. Each question specifies an answer type, such as yes/no or a choice among labelled options. We read each label's score from the model's answer-token probabilities. A restricted softmax over those tokens, optionally temperature-scaled, gives a distribution over the labels. Although the scores sum to one, they are not necessarily calibrated probabilities of the outcomes being assessed (Section 2.3).

The idea relates to *P(True)*-style self-evaluation [18] and to temperature scaling [19]. It differs from LLM-as-judge setups that generate a rationale and a verdict [20], and from encoder classifier heads trained on vulnerability datasets [21, 22, 23, 24, 25].

Several open-weight models are published for this use, all under Apache-2.0: Intern-Decision-4B (4.5B, fine-tuned by InternLM from Qwen3.5-4B) [17, 26]; clef (27B) [27]; pplx-decider-v1.1 (26B) [28]; and decider-chat-gemma4-31b (31B) [29]. Intern-Decision-4B is not Jev but an independent open-weight member of the same class: its card calls it a "structured decision model", its responses follow the Jev envelope, and it reports a mean of 44 ms per query against 110 ms for Jev on the same benchmark [17]. AnyJev turns any causal language model served by vLLM into such a decider by reading label log-probabilities [30, 31].

Two details matter in practice.

- **Read labels by id.** In one case a 31B model at temperature 1 put the "No" label below the top 20 tokens, so a fixed top-K read could not see it.
- **Scores are not calibrated at the real base rate.** In one evaluation a model's mean stated probability was 0.35 against a true rate of 0.005. Scores must rank, not decide.

2.2 Signals for vulnerability discovery

Revision history and current code both offer clues about potential vulnerabilities. Previous security fixes are one source of evidence: a patch may add a necessary check at one location while leaving a similar operation elsewhere unchanged, or add a check that fails to exclude every unsafe value. These situations motivate patch-based analysis and research on recurring vulnerabilities [32, 33, 34]. If a patch adds a length check to one function that copies input into a fixed-size buffer, a similar copy in another function may warrant investigation. The missing check marks a location worth analysing, although callers may constrain the second function's inputs; it does not by itself establish a vulnerability. We call a tool that finds such candidates in revision history an *incomplete-fix miner*.

Code patterns can also reveal candidates without a previous fix: input-dependent lengths used in buffer writes, arithmetic that can wrap before an allocation or copy, or externally supplied sizes passed to memory operations. We call a detector for one such pattern a *shape localizer*. Both tools propose locations. Establishing a vulnerability requires examining the surrounding constraints and execution paths.

Fuzzing coverage also helps us choose projects. Fuzzing runs software on generated or mutated inputs to expose failures, and memory-error detectors such as AddressSanitizer flag invalid accesses during those runs [35]. Projects without known fuzzing coverage are reasonable search targets, although the absence of documented coverage does not establish that a project has never been fuzzed, and existing coverage does not establish the absence of vulnerabilities.

2.3 Ranking, classification and calibration

A score can be used to classify candidates or to order them. Classification applies a threshold: candidates above it are labelled positive and those below negative. Ranking orders candidates by score so that higher-scoring ones are examined first. The two uses impose different requirements. A model may assign higher scores to promising candidates without supporting a reliable threshold for deciding whether any single function is vulnerable. Under a limited review budget, a useful ordering increases the number of real findings among the candidates read first; it does not establish the validity of any individual finding.

Calibration concerns whether predicted probabilities agree with observed frequencies [19]: among candidates assigned a probability near 0.8, a calibrated predictor is right about 80% of the time. Answer-token probabilities do not automatically have this property for vulnerability detection. The *base rate* is the share of genuine vulnerabilities in the population examined; when it is small, even a modest false-positive rate produces many more false alarms than real findings. Evaluation must therefore consider the candidate population and the review budget, not only the model's ability to separate positive from negative examples. Our harness therefore uses decision-model scores to order the work and relies on contextual analysis and reproduction to confirm findings.

3 Related work

Decision models for security (closest work). The closest prior work is CT0, a cybersecurity model by Mayoral-Vilches [36]. CT0-Micro, at roughly 4B parameters, pairs a "System 1" fast path that answers typed security decisions in a single forward pass without generating tokens with a "System 2" tool-using agent mode behind a confidence gate. It reports 21.1% of Cybench subtasks solved on the agentic path (against 7.6% for its base model), IOC file screening at about 80 files per second, jailbreak detection at AUROC 0.966, and vulnerability-fact extraction that beats either mode alone at about a fifth of the generative cost. CT0 shows that System-One-style decision models work for security classification and triage. Our work complements CT0 by applying an off-the-shelf model of this class to vulnerability *discovery* in real open-source code at scale. Our harness decides where to look and uses scores mainly to rank, keeping only a low reach-score floor on some queues; we report measured throughput and confirmed findings.

Theori's AIxCC system RoboDuck is the closest prior *design*: it ranked static-analysis candidates by the log-probabilities of a single "likely/unlikely" LLM classification and sent only the top fifth to a costly agent step [37]. Needles at Scale likewise uses a low-cost language model to rank structurally pre-filtered Windows binary functions for review [38]. Two results support reading labels from one pass rather than from generated text: in SOC alert triage, chain-of-thought reasoning degraded the label-token probabilities that automated triage relies on [39], and hidden states separate vulnerable from secure code better than prompting does [40].

Static analysis with learned triage. Z-Ranking proposed ranking a static analyser's reports rather than filtering them [41]. Heckman and Williams survey actionable-alert identification [42]; Kang et al. show how leakage and duplication inflated earlier learned results [43]. LLMs now inspect static bug warnings [44], cut false positives in industrial static bug detection [45], adjudicate CodeQL alerts with flow context [46] and adjudicate alerts with multi-run consistency checks [47]. IRIS infers taint specifications for CodeQL and filters its alerts [48], LLift triages use-before-initialisation candidates in the Linux kernel [49], and LLMDFA synthesises dataflow analyses [50]. Our design is closest to IRIS, LLift and RoboDuck: cheap analysis proposes candidates, and a model ranks them. Here, shape localizers and incomplete-fix miners take the place of a full SAST engine, and a single-pass decision model takes the place of a generative reasoner. Overconfidence in LLM security judgements is widespread, and confidence is best used as a relative triage signal [51], which is how we use it.

Small and efficient models for vulnerability detection. Researchers have specialised small generative models for detection with reasoning and agent scaffolds, including the 7B VulnLLM-R [52] and R2Vul's distilled structured reasoning [53]. Others have compressed models via linear probes [54]. Small fine-tuned models do well on synthetic data [55] and can be distilled for C/C++ [56]; for an 8B Llama-class model, fine-tuning matters more than prompting [57]. Several results caution that fine-tuning moves thresholds more than understanding: on Linux-kernel C, fine-tuning shifted the output threshold without changing the decision policy [58]; fine-tuned detectors show high false-positive rates on vulnerable/patched pairs [59]; and a classifier on classic code metrics matches state-of-the-art LLMs [60]. A leakage-controlled re-evaluation of pre-trained models updates the CodeBERT line [61, 21, 24]. Vul-RAG adds retrieved vulnerability knowledge [62, 63], and LLM4Vuln separates a model's reasoning from its context [64]. Our results support that cautious reading: we use the model to rank, not to decide.

Benchmarks and the vulnerable-versus-patched problem. Risse and Böhme report the closest result to our patched-version AUC: top function-level models cannot tell vulnerable functions from their patched versions [65]. Paired evaluation has since become standard: PairVul [62], JitVul [66] and VulnLLMEval [67] all report that models struggle

to separate a function from its fix. Function-level labels are often undecidable without calling context [68], precision collapses at realistic base rates [69], LLM vulnerability analysis leans on surface patterns [70], and fix-derived datasets carry substantial label noise [71, 72, 73]. PrimeVul [74], ReVeal [75], SecVulEval [76], CASTLE [77] and broad LLM evaluations [78, 79, 80] likewise paint a pessimistic picture of detection alone. ARVO provides reproducible C/C++ vulnerable/patched builds [81].

Incomplete fixes and recurring vulnerabilities. Our incomplete-fix miners build on earlier recurring-vulnerability work: vulnerable clone detection [82], patch-enhanced vulnerability signatures [32], and multi-stage filtering with taint analysis [33]; an independent benchmark finds the best of these reach F1 0.55 on harder clones [83]. Patch studies show why incomplete fixes matter: about 6% of publicly reported security patches needed supplementary patches [84], incomplete fixes in the Linux kernel share recognisable root causes [34], and many vulnerability fixes take several patches [85]. Big Sleep framed LLM bug-finding as variant analysis of previously fixed bugs [11].

Agentic discovery and LLM-assisted fuzzing. Project Naptime and Big Sleep established tool-equipped agents that verify their own findings [10, 11, 86]; CodeMender and Aardvark extend this to patching [87, 88, 89]. Such agents have also reached the scale of a whole operating-system kernel: OpenAI's run that produced 948 Linux privilege-escalation bugs was priced at about US$1.2 million in API credits [14]. The DARPA AI Cyber Challenge produced complete cyber-reasoning systems combining LLMs with fuzzing and static analysis [90, 91, 92, 93, 94], since made locally deployable [95] and extended to multi-agent discovery on OSS-Fuzz targets [96]. Searching over agent harness designs found browser zero-days with an open-weight model [97]. Related work includes LLM-written fuzz drivers [98, 99, 100, 101, 102] and agent benchmarks for reproduction and proofs of concept [103, 104, 105, 106]. These systems use generative models throughout; ours confines the model to one ranking pass.

Small models and "system over model" (concurrent, 2026). Within days of the Mythos announcement, Fort showed that small and open-weight models, given the isolated vulnerable function, often recognise the same bugs, and argued that the advantage lies in the system rather than the model [107, 108]. Independent reproductions found the same. Function-level slicing outperformed whole-file prompting for open and frontier models alike, and at function level an open model matched a frontier model [109]. Local open-weight models on one consumer GPU found the target bug but drowned it in false positives until a reachability stage was added [110]. Other studies reproduced or bounded these results [111, 112, 113]. Earlier, a frontier model called without scaffolding found a kernel zero-day at a signal-to-noise ratio of about 1:50 [114]; AI analyzers have since been credited with OpenSSL CVEs [115] and, by the curl maintainer, with a large batch of real fixes [116]. Pesoli et al. argue that the bottleneck moves from discovery to validation [117]. Our results support the "system over model" argument with a single-pass decision model used mainly for ranking.

Memory-safety bug finding. Memory-safety defects remain the largest class of serious vulnerabilities in C and C++ code [2, 3, 118]. Our oracle is AddressSanitizer [35, 119], with hardened builds using `_FORTIFY_SOURCE` and stack protectors [120, 121]. Coverage-guided fuzzing is the standard way to find these bugs [122, 123, 124], and OSS-Fuzz applies it continuously to about a thousand projects [125]. Our results suggest that projects outside OSS-Fuzz are where a static approach pays off.

4 Method: the harness

4.1 Overview

By *harness* we mean the whole system around the model. It turns a corpus into a ranked list of candidate functions. Project selection chooses repositories; cheap deterministic signals locate candidate functions within them; a small decision model ranks those candidates rather than accepting or rejecting them, apart from a low reach-score floor on some queues (Section 4.5). Nothing counts as a finding until it has been reproduced. We call the program that reproduces a suspected defect a *reproducer*.

4.2 Scope

The harness targets memory-safety defects in open-source C and C++ software that can be triggered through untrusted input, such as files, network messages or documents. The harness reads source code and, for fix mining, Git history. Its focus is memory accesses, buffer sizes and length calculations. Business logic, authorisation and other classes that need different reasoning fall outside its scope. The search stages—corpus selection, miners, localizers and decision model—use no frontier model. Stronger coding agents take part only in verification; we report its outcomes but do not evaluate that stage. The resource-efficiency claims therefore concern search and prioritisation, not the whole process. Function-level scoring limits what the model sees. A caller, set-up code or build configuration may constrain a candidate that looks unsafe locally; only later investigation can establish those constraints.

4.3 Where to look: project selection

Project selection aims at software whose security affects many downstream users. We tried three strategies.

- **Curated domains** (file-format, codec, document and network parsers) gave low and mostly minor yield after the first rounds.
- **GitHub stars** pulled in dead and hobby projects, because stars measure lifetime popularity rather than deployment.
- **Reach ranking** replaced stars with a reach score over 4,245 C/C++ repositories, built from Repology's cross-distribution packaging data [126], Debian reverse dependencies and popularity-contest counts, vcpkg/Conan/Homebrew presence and deps.dev, with a liveness filter.

Known fuzzing coverage provides a second criterion. We use OSS-Fuzz membership as an observable indicator [125] and treat absence from it as a selection signal, not as proof that a project was never fuzzed. The reach-ranked head is heavily fuzzed and well guarded, so the final campaign rounds drew a *fresh* corpus of widely packaged projects that are mostly outside OSS-Fuzz (Section 5.2).

4.4 Where to look: miners and localizers

Incomplete-fix miners read the full Git history. They find commits that add a protective check around an unsafe operation, then look for sibling sites without the check or for checks that fail to exclude the fatal value. A sibling found this way is a hypothesis, not a verdict: its inputs may be constrained elsewhere.

Shape localizers do not depend on fix history. Each is a cheap syntactic detector for one defect shape, such as a fixed-size buffer written with an input-derived length, arithmetic on a length that can wrap before a copy, or an untrusted size reaching a memory operation. A further deterministic test keeps only functions that perform an unsafe memory operation; unlike a model-based cut, this is an explicit part of the candidate criteria. This lets the localizers propose candidates even where no relevant fix exists.

4.5 The decision: two questions, used to rank

Each candidate function is scored by Intern-Decision-4B [17] with two typed questions: whether the code processes data from an untrusted source (the *reach* question), and a second question matched to its defect shape. The model reads the function's code and returns scores that order the list. Until the localizer pilots of 2026-10-07, every queue also dropped candidates with a reach score below 0.27 (Table 4). From the overnight run of 2026-10-07 onward, the fix-miner and fixed-buffer queues kept every candidate and only flagged low reach scores; the other localizer queues kept the 0.27 reach floor through the last scan round on 2026-10-09. No queue used a cutoff on the shape question after the pilots. A candidate kept in the queue is not necessarily read. The review budget determines how far down the list reviewers go; ranking determines what they read first.

4.6 From lead to finding

A ranked lead becomes a finding only after verification. Triage and proof used separate, stronger models and human review of the surrounding program context. The decision model did not perform those stages. Every finding had to reproduce as an AddressSanitizer crash in a reproducer and again on a hardened build, and a person signs off before anything is disclosed. A reproduced crash shows failure under the tested conditions. Its security relevance also depends on the real program, including whether untrusted input can reach the failing path. This paper does not evaluate the effectiveness or cost of this stage.

5 Experimental setup

5.1 Research questions

- **RQ1: Classification.** How well do small decision models and other small classifiers separate vulnerable functions from unrelated benign functions, and from their own patched versions? (Section 6.1)
- **RQ2: Prioritisation.** How much do the harness's deterministic signals and the model's ranking concentrate real findings at the top of a limited review budget, compared with random order? (Section 6.2)
- **RQ3: Recall.** Of known vulnerabilities in projects without any of our findings, how many does the harness place within a repository's review budget, and where are the rest lost? (Section 6.3)

- **RQ4: Throughput and cost.** How fast is the decision step on data-centre-free hardware, from a desk-side GB10 to a consumer gaming GPU, and how does the harness compare in cost and output with a rule engine? (Section 6.4)
- **RQ5: Campaign outcomes.** What findings, severities and review outcomes did the campaign produce? (Section 6.5)

5.2 Datasets and study populations

CVE function set. 91 vulnerable/patched function pairs from real CVE fixes, plus 246 benign functions from 22 repositories; 71 pairs and 200 benign functions are native C/C++. The vulnerable-versus-benign comparison measures separation from unrelated code; the vulnerable-versus-patched comparison measures separation from the repaired version of the same function. The campaign called this its calibration set, but AUC on it does not by itself establish calibration.

Method study. 339 functions from 82 repositories in the first search rounds, labelled by hand: 15 bug-site functions belonging to 8 confirmed bugs (one bug can span several functions), 301 hand-rejected false positives, and 23 functions with ambiguous labels that were recorded but not scored. We count a bug as kept if any of its sites survives; AUC compares the 8 bugs with the 301 false positives.

Ranker evaluation at deployment prevalence. 3,199 labelled C/C++ records (252 true positives, 2,529 false positives and 418 uncertain) plus a background of 3,000 unread candidates, reweighted to one real bug per 200 candidates; precision@25 is the reviewer budget. The incumbent model chose what was read, biasing the labels toward it.

Campaign queues. The 51 real ranked queues that contain at least one confirmed finding: 44,121 rows, of which 91 map to 78 distinct confirmed findings by repository and function.

Known-CVE recall set. 75 public memory-safety CVEs drawn at random (fixed seed) from NVD by a rule fixed in the selection script before its single run (Appendix A). None is in a project with one of our findings: projects were excluded by name match against the campaign's disclosure status sheet. Most of the 75 projects (66, 54 as the same GitHub repository) appear in the campaign's scan lists, but the campaign scanned them at their October 2026 heads, after these fixes; the recall run checks out each repository at the parent of its fix.

Campaign corpus. The reach-ranked corpus (built 2026-10-07) has 4,245 C/C++ repositories, 311 (7.3%) of them on OSS-Fuzz. The fresh corpus (2026-10-09) starts from 155,827 Repology projects packaged in at least two families and keeps 2,832 maintained C/C++ repositories within a size limit that are in neither the reach-ranked list nor any earlier scan; 1,178 of them expose an attack surface, and 1,000 were mined. OSS-Fuzz membership was checked through deps.dev for the 1,278 fresh repositories hosted on GitHub at the time: 6 (0.5%) are enrolled.

Campaign findings. We report the memory-safety findings the campaign confirmed after review, across all corpora it searched (Section 6.5).

5.3 Comparisons and procedures

Small-classifier comparison. Six small open models on the native CVE function set, one at a time on one GB10 under vLLM 0.30.1: Granite Guardian 3.1-2B, 3.2-5B and 3.3-8B [127, 128, 129], Phi-4-mini-instruct [130] and xLAM-2 1B and 3B [131], with Intern-Decision-4B as reference. Only Intern is a trained decision model; the others were read as P(Yes) from a chat prompt or their own guard template. These are configuration baselines rather than a controlled model-family comparison. Each pairs a model with one readout and prompt, sometimes off its native template; none was evaluated as a ranker on real queues. Llama Guard was not run: it is gated, and its largest version exceeds the host's memory headroom.

Method study and ranker evaluation. The method study compares combinations of deterministic tests, models and questions by how many known bugs they keep and how many hand-rejected candidates they remove. We tested thresholds leaving one bug out, but chose the final design with all 8 bugs in view. Its 8-of-8 retention is therefore optimistic; with n = 8, a one-bug difference is noise. The ranker evaluation measures precision@25 at the reweighted deployment prevalence.

Ranking ablation. We re-order each campaign queue by the deployed decision-model score, at random (10,000 seeds), and by a simple heuristic (the localizer's own site count). We count confirmed findings in the top k of each queue, pooled over queues.

Recall procedure. Each known-CVE repository is checked out at the parent of its fix, and the campaign's where-to-look stages run unchanged, including their per-repository caps; every queued row is then scored by the decision model. A CVE counts as found at k if a function the fix changed is in that repository's top k (Appendix A).

Rule-engine comparison and hardware portability. OpenSSL at its master commit of 2026-10-07, a large, heavily fuzzed C repository where the campaign produced no lead. On the same 16-vCPU x86 host we ran Opengrep v1.31.0 with the opengrep-rules C ruleset (16 rules, 3 runs, 16 jobs) and the harness's CPU stages, unchanged. The decision step (278 decisions) was timed on one GB10 under vLLM with batching; on a consumer RTX 4090 in a gaming PC with the same script and engine settings, after a 25-minute CPU and GPU burn-in with checked results; and on a shared laptop's integrated GPU. We measure time, resources and output volume. Neither output was triaged, and CodeQL was not run, so this comparison does not measure detection quality.

5.4 Metrics and uncertainty

ROC-AUC is the probability that a randomly chosen positive receives a higher score than a randomly chosen negative, with ties counted as half; 0.5 is chance and 1.0 perfect separation. We report vulnerable-versus-benign and vulnerable-versus-patched AUC separately. AUC is neither accuracy nor a measure of calibration.

Precision@k is the share of labelled positives among the first k ranked candidates. It measures the concentration of positives within a fixed review budget. We use budgets of 10, 25 and 40 and state the population and any reweighting with each result.

Recall and false-positive rate at a threshold are TP/(TP + FN) and FP/(FP + TN), over the labelled population evaluated. In the recall study, recall at k is the share of the 75 CVEs whose changed function is in the repository's top k. Neither establishes the recall of the whole campaign, whose missed vulnerabilities are unknown.

Review outcomes. The share of hand-rejected candidates a filter removes measures false-alarm reduction on that set, and the triage keep rate measures how many ranked leads survive triage. Their denominators differ, so they cannot be combined into one pipeline false-positive rate.

Throughput is given in decisions or function chunks per second, with the unit stated for each experiment; rows per second are not comparable across models that answer different numbers of questions with different prompt lengths. Timings exclude investigation and disclosure.

Uncertainty. For Intern-Decision-4B on the CVE function set we report 95% percentile intervals from a bootstrap that draws each vulnerable/patched pair as one unit and the benign functions separately (1,000 replicates, seed 20261110). The ranking ablation reports 95% intervals from a bootstrap over queues; the recall study, 95% intervals on each recall. These intervals describe uncertainty in individual estimates, not in the difference between models or methods.

5.5 Severity assessment

Severity is assessed with the Common Vulnerability Scoring System v3.1 [132], whose qualitative bands are none (0), low (0.1–3.9), medium (4.0–6.9), high (7.0–8.9) and critical (9.0–10.0). Each confirmed vulnerability is rated by worst-case impact from its own write-up, with one rubric applied by independent raters, and a blind second rating of 20 findings. Memory-corrupting writes take the code-execution ceiling, except weak writes (a single byte, attacker-uncontrolled bytes, small values inside one structure), which are scored at low confidentiality and integrity impact unless the attacker chooses the offset; out-of-bounds reads and crashes never reach critical. Each rating accounts for reachability from untrusted input. These are campaign scoring assumptions, not general rules implied by a defect class. A reproduced crash demonstrates failure; the assumed impact may go beyond what was demonstrated.

5.6 Implementation and hardware

Decision hosts. NVIDIA DGX Spark desktop systems (GB10 Grace Blackwell): one GPU, 20 Arm cores and 119.7 GiB of memory shared between CPU and GPU. The campaign used three. NVIDIA's list price for the 128 GB DGX Spark Founders Edition was US$3,999 at its October 2025 launch and US$4,699 from February 2026 [133, 134]; by October 2026 it was listed at US$6,950 [133]. We used DGX Spark Founders Edition units, so these list prices describe our hardware. They do not measure campaign cost. Overcommitting unified memory hangs the host rather than failing a request. We therefore loaded models strictly in sequence behind a free-memory guard.

Serving. Throughput experiments used vLLM with batching (a 0.30.1 development build on the GB10; 0.32.0 on the RTX 4090). The 4B ranker's weights are about 8.5 GB in BF16; total runtime memory also depends on input lengths and serving settings.

Build host. Builds and sanitizer runs used an x86-64 build server.

Commodity hardware. We ran the unchanged ranker on a gaming PC's consumer GPU (NVIDIA RTX 4090, 24 GB, under WSL2) and on a laptop's integrated GPU (Apple M4 Max, without vLLM), and compared its scores with the GB10's (Section 6.4).

6 Results

6.1 RQ1: Classification

Alone, a small decision model is a weak detector. On the native C/C++ function set, Intern-Decision-4B separates vulnerable from benign functions with AUC 0.73, but a vulnerable function from its own patch with only 0.56. At a fixed threshold of 0.5 under vLLM, it recalls 15% of vulnerable functions at a 6% benign false-positive rate. An earlier campaign run used a different serving stack on the same functions and gave 20% recall and 7% benign false-positive rate at that threshold, with the same AUCs (0.732 and 0.566). The ranking was stable, but even one model's threshold behaviour moved with how it was served. The campaign's other candidate deciders were within a few points of it (Table 1). Against patched code (all 91 pairs), all lie between 0.55 and 0.57; these point estimates do not establish a reliable ordering.

Table 1: Campaign deciders on the CVE function set. A dash marks a measurement not taken. DiffusionGemma-26B-A4B [135] was run as NVIDIA's NVFP4 build [136]. The patched column covers all 91 pairs, including 20 in managed languages; on the 71 native pairs Intern scores 0.564 (Table 2).

Configuration	AUC vs benign (native C/C++)	AUC vs patched (all 91 pairs)	Throughput on one GB10
4B + 27B two-tier (rank mean)	0.76	0.57	—
clef-27B	0.74	0.56	0.45 chunks/s
Intern-Decision-4B · memory-safety question	0.73	0.55	4.7 chunks/s
DiffusionGemma-26B-A4B	0.69	0.55	—

Six configuration baselines, measured. The comparison concerns scoring configurations rather than model families (Section 5.3). None separates a vulnerable function from its patch better (Table 2): AUC against the patched version ranged from 0.50 to 0.62. Against benign code, xLAM-2-1B (0.77) and Intern (0.73) scored highest. Intern's own 95% interval is 0.66–0.79 against benign code and 0.52–0.61 against patched code (pair-preserving bootstrap, Section 5.4). xLAM-2-1B's 0.77 lies within the first interval. This describes the uncertainty in Intern's estimate on 71 vulnerable functions; it is not a test of the difference between the two models. The interval comes from the campaign's scores for the same model and functions (AUC 0.732 and 0.566). Table 2 shows the later vLLM run, whose point estimates agree to 0.002. The threshold of 0.5 behaved very differently across models: Granite Guardian 3.1-2B flagged 84% of benign code, while xLAM-2-3B flagged nothing. Table 3 gives sizes, readouts and throughput.

Table 2: Small classifiers on the native C/C++ function set (71 vulnerable, 71 patched, 200 benign), one GB10 under vLLM. Intern's 95% intervals are given in the text.

Model	AUC vs benign	AUC vs patched	Recall at 0.5	Benign FPR at 0.5
Intern-Decision-4B	0.731	0.564	15%	6%
xLAM-2-1b-fc-r	0.773	0.590	73%	32%
xLAM-2-3b-fc-r	0.688	0.544	0%	0%
Phi-4-mini-instruct	0.666	0.617	79%	50%
Granite Guardian 3.1-2B	0.672	0.555	99%	84%
Granite Guardian 3.2-5B	0.561	0.504	34%	23%
Granite Guardian 3.3-8B	0.599	0.589	17%	19%

Table 3: Configurations and throughput in the small-classifier comparison (one GB10, vLLM 0.30.1). *Intern answers two fields per row with longer prompts, so its rows/s is not comparable.

Model	Parameters	Readout	Rows/s
Intern-Decision-4B	4.5B	decision readout	4.7*
xLAM-2-1b-fc-r	1.5B	Yes/No	49.1
xLAM-2-3b-fc-r	3.1B	Yes/No	25.3
Phi-4-mini-instruct	3.8B	Yes/No	23.1
Granite Guardian 3.1-2B	2.5B	guard template	18.8
Granite Guardian 3.2-5B	5.8B	guard template	10.0
Granite Guardian 3.3-8B	8.2B	guard template	7.1

Why this ranker. In the method study, the 4B model had the *lowest* best-question AUC of four models (0.653 against 0.728–0.749). Yet its reach question was the only model question that kept all 8 bugs out of sample; a 27B alternative cost about ten times more per chunk and ranked no better inside the filtered pool.

6.2 RQ2: Prioritisation

Candidate yield in the localizer pilots. The pilots describe candidate yield by configuration (Table 4); they do not isolate the effect of ranking. For the length-wrap localizer, a random sample of 30 candidates held 2 leads (6.7%). The top 30 after ranking held 9 leads and 1 near-lead (30–33%). For the untrusted-size localizer, the figures were 3.3% and 24–36% in the top 25. For the fixed-buffer localizer, a random sample of 40 from its first version held 1 lead (2.5%); the top 40 of its revised version held 14 leads and 3 near-leads (35–42.5%). That list includes the repository used to validate the localizer. Without it, 4 of 24 (16.7%) were leads. These shares measure the queue at triage, not final findings. A random sample is not a random ordering at the same budget. The columns also differ in localizer version, deterministic test and reach cut, so we reserve *ranking lift* for the within-queue comparisons below.

Table 4: Candidate yield in the localizer pilots, by configuration, at triage (descriptive). Raw rates are hit rates in a random sample of the localizer's candidates; ranked rates are hit rates in the top k after the deterministic test, a reach cut at 0.27 and ranking by the shape question. Ranges count leads only and leads plus near-leads; they are not confidence intervals.

Candidate source	Random sample	Raw rate	Top k after ranking	Ranked rate
Fixed-buffer localizer	40 (first version)	2.5%	top 40 (revised version)	35–42.5% (16.7% without the validation repository)
Length-wrap localizer	30	6.7%	top 30	30–33%
Untrusted-size localizer	30	3.3%	top 25	24–36%

Ranking lift against random order. The recall study provides a cleaner check because its labels come from public CVE fixes (Section 6.3). Within the same queues, the model placed the fixed functions in the top 10, 25 and 40 for 9, 16 and 17 CVEs, against 7.0, 11.1 and 13.4 expected at random: a lift of 1.3–1.4 times. On the campaign's own queues the observed lift is larger. Across the 51 campaign queues with a confirmed finding, the model's order put 24 confirmed findings in the top 10, against 3.2 expected at random and 6 for the site-count heuristic. That is a 7.5-fold lift (95% CI 4.3–14.1). At the top 25 the counts were 32, 6.7 and 13 (4.8-fold, CI 2.9–8.0); at the top 40, 37, 9.5 and 23. Confirmed findings sat at a mean percentile rank of 0.14 under the model, 0.37 under the heuristic and 0.5 at random. The observed lift is a selection-biased upper bound. The 51 queues were chosen because they contain a confirmed finding, reviewers read from the top of these same rankings, and rejected rows rank even higher (0.12).

One question after one free test. In the method study every model reached AUC 0.65–0.75, and cascades of model judgements lost bugs (the best two-stage cascade kept 3 of 8). The best design placed a cheap model after a free deterministic unsafe-operation test and asked one reach question. It cut the hand-rejected false alarms from 301 to 132 (-56%) and kept all 8 bugs.

Ranking, not thresholding. The same model failed repeatedly as a gate. A cutoff at 0.5 kept none of three raw fixed-buffer hits; a rounded published threshold silently dropped 2 of 8 known bugs; and inside a band already above 0.7 the score carried almost no information (mean 0.7782 for kept leads, 0.7684 for discarded ones).

Where to look matters most. After triage, the under-fuzzed corpus kept 20.1% of ranked leads with reach $\geq$0.7, against 4–12% for the reach-ranked rounds. The incomplete-fix class (a guard added in one place and missed in its twin) was the most reliable source of real findings. Mature, heavily fuzzed code (cryptography, DNS, fonts, the most depended-on libraries) yielded nothing in our campaign. This describes our chosen targets, not those categories in general.

At the deployment base rate. At a reweighted prevalence of one real bug per 200 candidates, the 4B model reached precision@25 of 0.092, versus 0.062 for a 27B alternative. Fusing the two scored 0.26, but the gain was not proven at a 25–50-read budget, so we did not adopt it.

6.3 RQ3: Recall on known CVEs

On 75 public CVEs from projects without any of our findings (Table 5), the fixed function reached its repository's top 10 for 9 (12%), the top 25 for 16 (21%) and the top 40 for 17 (23%). Almost all of the loss happens before ranking. The changed function appeared in the chunked corpus for 72 of 75 CVEs. A localizer or miner flagged it for only 29, and 18 survived the per-repository caps to be scored. Once scored, the model placed 16 of 18 in the top 25. Random order would place about 11 in the top 25 of these short queues (a median of 60 functions per repository), so ranking adds less here than on the campaign's long queues. The localizers target specific write shapes. They flagged 18 of 41 out-of-bounds writes, but only 7 of 19 out-of-bounds reads and 2 of 8 use-after-free defects.

Table 5: Recall on 75 known CVEs. 95% intervals for recall at 10, 25 and 40: 5–20%, 12–31% and 13–32%.

Stage (n = 75 CVEs)	Count	Share
Changed function in the chunked corpus	72	96%
Flagged by a localizer or miner (before caps)	29	39%
Scored (survived the per-repository caps)	18	24%
In the repository's top 10 / 25 / 40	9 / 16 / 17	12% / 21% / 23%
Expected at random, same queues (top 10 / 25 / 40)	7.0 / 11.1 / 13.4	

6.4 RQ4: Throughput and cost

A desk-side GB10. With vLLM batching on one GB10, Intern-Decision-4B made 278 decisions in 63 s (4.4 decisions per second, 8.0 GiB of weights). During the campaign it scored 3.5–5.5 function chunks per second at 95% GPU utilisation, against 0.45–0.89 for a dense 27B decider.

A consumer gaming PC. On an RTX 4090 in a gaming PC (24 GB, under WSL2), the same script made the same 278 OpenSSL decisions in 25–29 s: 9.8 decisions per second on the slowest of three passes, 2.2 times the GB10, after a 97 s cold start. The scores matched the GB10's closely: 275 of 278 were within 0.02, all 278 were on the same side of 0.5, and 277 of 278 CWE choices were identical. The run used vLLM 0.32.0. Device memory peaked at 15.2 GiB, including the desktop's own use; a 25-minute burn-in beforehand produced no mismatches or hardware errors. The decision step thus ran faster on the gaming PC than on the GB10.

Even a laptop. On a shared laptop's integrated GPU without vLLM, the same decision step took about 10 minutes; 270 of 278 scores were within 0.02 of the GB10's. This is a floor, not the harness's speed.

Scale. All 4,245 reach-ranked repositories were scanned by both the fix miners and the shape localizers by 2026-10-09. Mining ran at 900–1,100 repositories per hour across two systems; one 300-repository batch took 18 minutes to mine and 4 minutes to score. These timings cover the search stages only.

Against a rule engine. On one repository, Opengrep is about ten times faster end to end (Table 6). Nearly all of the gap comes from the harness's single-process CPU stages, which the campaign parallelised across repositories rather than within one. The GPU step took about one minute on the GB10 and under half a minute on the RTX 4090. Their outputs also differ. One hygiene rule dominated Opengrep's 554 warnings (`memset` that may be optimised away: 468),

while the harness produced a ranked list of 139 rows. They also looked mostly at different code: only 33 of Opengrep's warnings fall inside the 121 functions the harness sent to the model.

Table 6: One-repository cost comparison on OpenSSL. CPU stages ran on the same 16-vCPU x86 host. The harness's 139 ranked rows cover 121 distinct functions. Neither output was triaged, so this is not a quality comparison.

	Opengrep v1.31.0, C rules	Harness + 4B decision model
CPU wall time (same 16-vCPU host)	27–31 s at 16 jobs	≈268 s, single process
Decision step, one GB10, vLLM	—	278 decisions in 63 s (4.4/s)
Decision step, consumer RTX 4090, vLLM	—	278 decisions in 25–29 s (9.8/s)
Decision step, laptop integrated GPU (floor)	—	278 decisions in ≈10 min
End to end	≈0.5 min	≈5.5 min (GB10); ≈5 min (RTX 4090)
Output	554 warnings in 227 files	139 ranked rows; 5 where both questions agree

Context: the cost of frontier-scale search. For context, OpenAI reported a single run of a cyber-specialised frontier model against the Linux kernel that produced 948 local privilege-escalation bugs. It gives the cost as US$1,213,430 in API credits, or about US$1,280 per bug, and describes that figure as an upper bound [14]. Both efforts look for exploitable defects in C systems code, but they spend compute differently. OpenAI's figure covers a frontier model doing search, validation and exploit development. In our harness, search and ranking run on three desk-side systems with a list price of US$4,000–7,000 each, or on a gaming PC; only the read candidates go to stronger models. The efforts are not comparable per finding. Their targets differ (one kernel versus thousands of user-space packages), as does the bar for a finding (a working root exploit versus a reproduced memory-safety crash); we also do not measure the cost of our verification stage. We have no end-to-end cost accounting. Total model-agent spend, verification cost and reviewer time were not recorded, and our GPUs are owned hardware with no metered cost. Whether a cheap search stage lowers the total cost of finding a vulnerability is therefore a hypothesis this paper motivates but does not test.

6.5 RQ5: Campaign outcomes

Findings and severity. The campaign confirmed 187 memory-safety findings. Each reproduced as an AddressSanitizer crash in a reproducer and again on a hardened build, and each was accepted in review (Section 4.6); reproduction in a reproducer does not by itself show reachability through a project's shipped entry point. Table 7 gives their worst-case CVSS v3.1 ratings. Under the rubric of Section 5.5, 160 of the 187 (86%) are rated critical or high. All 12 critical ratings are memory-corrupting writes reachable without authentication from the network or from clients that run unattended. The blind second rating agreed on all 20 findings it covered. These ratings assume worst-case impact; they do not establish demonstrated code execution for every finding.

Table 7: Worst-case CVSS v3.1 ratings of the 187 confirmed findings.

Worst-case CVSS v3.1	Score range	Confirmed findings
Critical	9.0–10.0	12
High	7.0–8.9	148
Medium	4.0–6.9	23
Low	0.1–3.9	4
Total		187

False positives. The record offers several measures, each with a different denominator. They cannot be combined into one false-positive rate.

- **Candidate yield (triage queue, localizer pilots; descriptive):** 6.7% → 30–33% in the top 30 (length wrap), 3.3% → 24–36% in the top 25 (untrusted size), and 2.5% → 35–42.5% in the top 40 (fixed buffer; 16.7% without its validation repository).

- **Deterministic test plus one question (method study):** hand-rejected false alarms 301 → 132 (−56%), all 8 known bugs kept.
- **Classification (known CVEs):** at a fixed 0.5, recall 15% at a 6% benign false-positive rate under vLLM (20% and 7% in the campaign's own run of the same model), which is why scores are used to rank rather than to decide.
- **Triage keep rate:** 20.1% of leads with reach ≥0.7 kept in the under-fuzzed corpus, against 4–12% in reach-ranked rounds.

What is hard. Per-function questions miss context outside the function: a caller's check, a build flag that compiles the code out, or a value fixed by set-up code. Applied to smart contracts, the same approach produced no confirmed leads (61 of 61 false positives); real exploits there need cross-function reasoning. Business logic, authorisation and cross-function flows remain outside what a single-function decision model does well.

7 Responsible disclosure and ethics

This paper names no project, function or defect affected by our findings; the appendix lists only public CVEs, already fixed, used to measure recall. All work ran in contained local builds. Nothing is sent without human sign-off, and the operator declined to report findings in one project of unclear legitimacy.

Disclosure. All confirmed findings are being disclosed to maintainers under coordinated disclosure before publication. We report findings project by project through each project's security channel and work with maintainers on fixes. We will share details of some findings once fixes are available. Until then, we report no per-project or per-fix status.

Dual use. These methods lower the cost of finding bugs. We publish the design, but withhold the corpora, lead lists, harnesses and proofs of concept: maintainers and defenders benefit from knowing that this kind of search is now cheap.

8 Limitations

- **Triage and proof used stronger models.** The corpus, miners, localizers and decision model use no frontier model. Triage and proof used separate, stronger coding-agent models; we did not measure their cost. A fully small-model pipeline is untested.
- **Recall is bounded by where to look.** On known CVEs, the harness found about one in five within a repository's top 25. Most misses were never flagged by a localizer. The localizers cover specific write shapes. Reads, use-after-free and logic errors need further localizers; larger per-repository caps would recover some flagged functions (11 of 75 in the recall study).
- **The ranking ablation is an upper bound.** Reviewers read from the model's own ranking, making low-ranked findings less likely to be confirmed at all.
- **Small samples and biased labels.** The method study has 8 bugs, and the ranker evaluation's labels were chosen by the incumbent model.
- **Small-classifier comparison.** Only Intern is a trained decision model, and the other six models were adapted. The sample contains 71 vulnerable functions; ranking on real queues was measured for Intern only.
- **Only a cost comparison with rule engines,** on one repository; detection quality against Opengrep or CodeQL is future work.
- **Scope.** C/C++ memory safety only.

9 Conclusion

For the model and harness we studied, a small, fast decision model helped find vulnerabilities at scale because the harness around it did the searching. On its own, the 4B model barely tells a vulnerable function from its fix. The harness decides where to look with incomplete-fix miners, shape localizers and a corpus chosen for reach and for the absence of fuzzing; it uses the model to rank rather than to decide, apart from a low reach-score floor on some queues. For known CVEs from projects without any of our findings, the model placed fixed functions within the review budget 1.3–1.4 times as often as random order of the same queues. On the campaign's queues it put 7.5 times more confirmed findings in the top 10 than random order, an observed upper bound. Across three desk-side GPU systems, the harness searched more than 4,000 projects, and the campaign confirmed 187 memory-safety findings. The search needed no frontier model, and its decision step runs at 9.8 decisions per second on a consumer gaming GPU. On known CVEs, recall is bounded mainly by where the harness looks. That is where the next gains lie. Our interpretation, which this

study does not test as a controlled comparison of model sizes, is that careful harness design, more than model size, is what makes early, large-scale vulnerability finding affordable for defenders and maintainers.

10 Acknowledgements

We thank Cyrex for supporting this research and NVIDIA for the GB10 (DGX Spark-class) platform used in the experiments. We thank the open-source maintainers who triaged, discussed and fixed our reports; we do not name them until their fixes are public. We thank the InternLM team for releasing Intern-Decision, the Qwen team for the base models it builds on, and the vLLM project for the open serving stack the experiments ran on. The corpus selection relied on public data from Repology, the Debian popularity contest and the OSS-Fuzz project list.

References

[1] László Szekeres, Mathias Payer, Tao Wei, and Dawn Song. SoK: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62. IEEE, 2013. doi: 10.1109/SP.2013.13. Added from co-author review (Gkogkou, 2026-11-02).

[2] Gavin Thomas. A proactive approach to more secure code. Microsoft Security Response Center blog, July 2019. URL https://www.microsoft.com/en-us/msrc/blog/2019/07/a-proactive-approach-to-more-secure-code. Posted 2019-07-16; " 70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues".

[3] The Chromium Projects. Memory safety (Chromium security), n.d. URL https://www.chromium.org/Home/chromium-security/memory-safety/. 70% of high-severity Chromium security bugs are memory-safety bugs (912 bugs since 2015). Undated page; accessed 2026-10-28.

[4] Semgrep, Inc. Semgrep docs, 2026. URL https://docs.semgrep.dev/.

[5] Opengrep contributors. Opengrep: static code analysis engine to find security issues in code, 2025. URL https://github.com/opengrep/opengrep. LGPL-2.1 fork of Semgrep v1.100.0, announced 2025-01-23.

[6] GitHub. CodeQL documentation, 2026. URL https://codeql.github.com/docs/.

[7] Pavel Avgustinov, Oege de Moor, Michael Peyton Jones, and Max Schäfer. QL: Object-oriented queries on relational data. In *30th European Conference on Object-Oriented Programming (ECOOP 2016)*, volume 56 of *LIPIcs*, pages 2:1–2:25, 2016. doi: 10.4230/LIPIcs.ECOOP.2016.2. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2016.2.

[8] Stephan Lipp, Sebastian Banescu, and Alexander Pretschner. An empirical study on the effectiveness of static C code analyzers for vulnerability detection. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 544–555, 2022. URL https://mediatum.ub.tum.de/doc/1659728/1659728.pdf. ISSTA program lists title as "...Static C/C++ Analyzers...".

[9] Wachiraphan Charoenwet, Patanamon Thongtanunam, Van-Thuan Pham, and Christoph Treude. An empirical study of static analysis tools for secure code review. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 691–703, 2024. doi: 10.1145/3650212.3680313. URL https://arxiv.org/abs/2407.12241.

[10] Sergei Glazunov and Mark Brand. Project naptime: Evaluating offensive security capabilities of large language models. Google Project Zero blog, June 2024. URL https://projectzero.google/2024/06/project-naptime.html. Posted 2024-06-20.

[11] Big Sleep team (Google Project Zero and Google DeepMind). From naptime to big sleep: Using large language models to catch vulnerabilities in real-world code. Google Project Zero blog, November 2024. URL https://projectzero.google/2024/10/from-naptime-to-big-sleep.html. Posted 2024-11-01; SQLite seriesBestIndex stack buffer underflow.

[12] Anthropic. Project Glasswing: Securing critical software for the AI era. Anthropic announcement, April 2026. URL https://www.anthropic.com/glasswing. Announced 2026-04-07.

[13] Nicholas Carlini, Newton Cheng, Keane Lucas, Michael Moore, Milad Nasr, Vinay Prabhushankar, Winnie Xiao, et al. Assessing Claude Mythos Preview's cybersecurity capabilities. Anthropic Frontier Red Team blog, April 2026. URL https://www.anthropic.com/research/mythos-preview. Posted 2026-04-07 (formerly red.anthropic.com/2026/mythos-preview).

[14] OpenAI. A thousand fewer paths to root. OpenAI research blog, November 2026. URL https://openai.com/index/a-thousand-fewer-paths-to-root/. Published 2026-11-04.

[15] OpenAI. GPT-6 Cyber: Advancing the frontier of aligned cybersecurity agents. OpenAI announcement, November 2026. URL https://openai.com/index/introducing-gpt-6-cyber/. Published 2026-11-05.

[16] TypeSafe AI. Jev: a system one model. https://typesafe.ai, 2026. First public "System One" model: typed questions in, typed decisions with probabilities out; API early access. Announced September 2026 (press: 2026-09-15).

[17] InternLM. Intern-Decision-4B. Hugging Face model card, https://huggingface.co/internlm/Intern-Decision-4B, 2026. 4.54B parameters (safetensors total), base model Qwen/Qwen3.5-4B, Apache-2.0, published 2026-09-26; Jev-compatible responses, benchmarked against Jev; technical report arXiv:2610.22131.

[18] Saurav Kadavath et al. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221. 36 authors.

[19] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, pages 1321–1330, 2017. URL https://arxiv.org/abs/1706.04599.

[20] Lianmin Zheng et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *NeurIPS Datasets and Benchmarks Track*, pages 46595–46623, 2023. URL https://arxiv.org/abs/2306.05685. 13 authors.

[21] Zhangyin Feng et al. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of EMNLP*, pages 1536–1547, 2020. doi: 10.18653/v1/2020.findings-emnlp.139. URL https://arxiv.org/abs/2002.08155. 11 authors.

[22] Daya Guo et al. GraphCodeBERT: Pre-training code representations with data flow. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2009.08366. 18 authors.

[23] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. UniXcoder: Unified cross-modal pre-training for code representation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7212–7225, 2022. doi: 10.18653/v1/2022.acl-long.499. URL https://arxiv.org/abs/2203.03850.

[24] Michael Fu and Chakkrit Tantithamthavorn. LineVul: A transformer-based line-level vulnerability prediction. In *IEEE/ACM International Conference on Mining Software Repositories (MSR)*, pages 608–620, 2022. URL https://ieeexplore.ieee.org/document/9796256.

[25] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://arxiv.org/abs/1909.03496.

[26] Qwen Team. Qwen3.5-4B. Hugging Face model card, https://huggingface.co/Qwen/Qwen3.5-4B, 2026. 4.66B parameters, Apache-2.0.

[27] Cloudflare. clef. Hugging Face model card, https://huggingface.co/Cloudflare/clef, 2026. 27.4B parameters, base model Qwen/Qwen3.8-27B, Apache-2.0, published 2026-09-30.

[28] Perplexity AI. pplx-decider-v1.1-27b. Hugging Face model card, https://huggingface.co/perplexity-ai/pplx-decider-v1.1-27b, 2026. 26.1B parameters, base model Qwen/Qwen3.8-27B, Apache-2.0, published 2026-10-05.

[29] Mapika. decider-chat-gemma4-31b. Hugging Face model card, https://huggingface.co/Mapika/decider-chat-gemma4-31b, 2026. 31.3B parameters, base model google/gemma-4-31B-it, Apache-2.0, published 2026-09-29.

[30] Jiamu Zhang, Tianze Yang, Yucheng Shi, and Liang Wu. AnyJev: Turn any LLM into a jev-style decision model. Nokia Applied Research, https://github.com/nokia-applied-research/AnyJev; PyPI anyjev 0.3.0, 2026. Apache-2.0.

[31] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

[32] Yang Xiao, Bihuan Chen, Chendong Yu, Zhengzi Xu, Zimu Yuan, Feng Li, Binghong Liu, Yang Liu, Wei Huo, Wei Zou, and Wenchang Shi. MVP: Detecting vulnerabilities using patch-enhanced vulnerability signatures. In *29th USENIX Security Symposium*, pages 1165–1182, 2020.

[33] Siyue Feng, Yueming Wu, Wenjie Xue, Sikui Pan, Deqing Zou, Yang Liu, and Hai Jin. FIRE: Combining multi-stage filtering with taint analysis for scalable recurring vulnerability detection. In *33rd USENIX Security Symposium*, pages 1867–1884, 2024.

[34] Qiang Liu, Wenlong Zhang, Muhui Jiang, Lei Wu, and Yajin Zhou. Characteristics, root causes, and detection of incomplete security bug fixes in the Linux kernel, 2025.

[35] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. AddressSanitizer: A fast address sanity checker. In *USENIX Annual Technical Conference (USENIX ATC)*, pages 309–318, 2012. URL https://www.usenix.org/conference/atc12/technical-sessions/presentation/serebryany.

[36] Víctor Mayoral-Vilches. CT0: Cybersecurity transformer, 2026. arXiv:2610.16550v2, 15 Oct 2026 (v1 12 Oct 2026).

[37] Xint (Theori). AI cyber challenge and Theori's RoboDuck. Theori blog, https://theori.io/blog/aixcc-and-roboduck-63447, 2025. 2025-08-08.

[38] Michael J. Bommarito, II. Needles at scale: LLM-assisted target selection for windows vulnerability research, 2026.

[39] Amol Khanna, Manu Nandan, Cristian Viorel Popa, Joan Pujol-Roig, Diana Bolocan, Laura Vasilie, Alexandru Apostu, Chase Helwig, Mihaela Gaman, Michael Brautbar, Edward Raff, Chase Midler, and Sven Krasser. Cybersecurity detection classification with reasoning-enabled language models, 2026.

[40] Weichen Yu, Ravi Mangal, Terry Zhuo, Matt Fredrikson, and Corina S. Pasareanu. A mixture of linear corrections generates secure code, 2025.

[41] Ted Kremenek and Dawson Engler. Z-ranking: Using statistical analysis to counter the impact of static analysis approximations. In *Static Analysis (SAS 2003)*, LNCS, pages 295–315, 2003. doi: 10.1007/3-540-44898-5_16.

[42] Sarah Heckman and Laurie Williams. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology*, 53(4):363–387, 2011. doi: 10.1016/j.infsof.2010.12.007.

[43] Hong Jin Kang, Khai Loong Aw, and David Lo. Detecting false alarms from automatic static analysis tools: How far are we? In *Proc. 44th International Conference on Software Engineering (ICSE)*, pages 698–709, 2022. doi: 10.1145/3510003.3510214.

[44] Cheng Wen, Yuandao Cai, Bin Zhang, Jie Su, Zhiwu Xu, Dugang Liu, Shengchao Qin, Zhong Ming, and Cong Tian. Automatically inspecting thousands of static bug warnings with large language model: How far are we? *ACM Transactions on Knowledge Discovery from Data*, 18(7):1–34, 2024. doi: 10.1145/3653718.

[45] Xueying Du, Jiayi Feng, Yi Zou, Wei Xu, Jie Ma, Wei Zhang, Sisi Liu, Xin Peng, and Yiling Lou. Reducing false positives in static bug detection with LLMs: An empirical study in industry. In *Proc. 48th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 716–727, 2026. doi: 10.1145/3786583.3786910.

[46] Mohsen Iranmanesh, Sina Moradi Sabet, Sina Marefat, Ali Javidi Ghasr, Allison Wilson, Iman Sharafaldin, and Mohammad A. Tayebi. ZeroFalse: Improving precision in static analysis with LLMs, 2025.

[47] William Klieber, David Svoboda, Lori Flynn, and Ruben Martins. Using LLMs to adjudicate static-analysis alerts with error reduction techniques, 2026.

[48] Ziyang Li, Saikat Dutta, and Mayur Naik. IRIS: LLM-assisted static analysis for detecting security vulnerabilities. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/582d4e27fa24168f3af1f4582655034b-Abstract-Conference.html.

[49] Haonan Li, Yu Hao, Yizhuo Zhai, and Zhiyun Qian. Enhancing static analysis for practical bug detection: An LLM-integrated approach. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):474–499, 2024. doi: 10.1145/3649828. URL https://2024.splashcon.org/details/splash-2024-oopsla/18/Enhancing-Static-Analysis-for-Practical-Bug-Detection-An-LLM-Integrated-Approach. LLift. arXiv preprint titled "The Hitchhiker's Guide to Program Analysis: A Journey with Large Language Models".

[50] Chengpeng Wang, Wuqi Zhang, Zian Su, Xiangzhe Xu, Xiaoheng Xie, and Xiangyu Zhang. LLMDFA: Analyzing dataflow in code with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 131545–131574, 2024. URL https://arxiv.org/abs/2402.10754. NeurIPS 2024 poster (per arXiv comment).

[51] Mohammed Latif Siddiq, Md. Nafiu Rahman, and Joanna C. S. Santos. An empirical study of security calibration in large language models for code. In *Proc. 42nd International Conference on Software Maintenance and Evolution (ICSME)*, pages 380–392, 2026. doi: 10.1109/ICSME67971.2026.00041. arXiv:2606.31159.

[52] Yuzhou Nie, Hongwei Li, Chengquan Guo, Ruizhe Jiang, Zhun Wang, Bo Li, Dawn Song, and Wenbo Guo. VulnLLM-R: Specialized reasoning LLM with agent scaffold for vulnerability detection, 2025.

[53] Martin Weyssow, Chengran Yang, Junkai Chen, Ratnadira Widyasari, Ting Zhang, Huihui Huang, Huu Hung Nguyen, Yan Naing Tun, Tan Bui, Yikun Li, Ang Han Wei, Frank Liauw, Eng Lieh Ouh, Lwin Khin Shar, and David Lo. R2Vul: Learning to reason about software vulnerabilities with reinforcement learning and structured reasoning distillation, 2025.

[54] Luis Ibanez-Lissen, Lorena Gonzalez-Manzano, Jose Maria de Fuentes, and Nicolas Anciaux. LPASS: Linear probes as stepping stones for vulnerability detection using compressed LLMs. *Journal of Information Security and Applications*, 93:104125, 2025. doi: 10.1016/j.jisa.2025.104125.

[55] Md. Azizul Hakim Bappy, Hossen A Mustafa, Prottoy Saha, and Rajinus Salehat. Case study: Fine-tuning small language models for accurate and private CWE detection in Python code, 2025. URL `https://arxiv.org/abs/2504.16584`.

[56] Zhiyuan Wei, Xiaoxuan Yang, Jing Sun, and Zijian Zhang. Distilling lightweight language models for C/C++ vulnerabilities, 2025. URL `https://arxiv.org/abs/2510.06645`.

[57] Dyna Soumhane Ouchebara and Stéphane Dupont. Llama-based source code vulnerability detection: Prompt engineering vs fine tuning. In *ESORICS*, pages 289–308, 2025. doi: 10.1007/978-3-032-07884-1_15. URL `https://arxiv.org/abs/2512.09006`. ESORICS 2025 acceptance per arXiv page.

[58] Arastoo Zibaeirad and Marco Vieira. Calibration without comprehension: Diagnosing the limits of fine-tuning LLMs for vulnerability detection in systems software, 2026.

[59] Feiyang Huang, Yuqiang Sun, Fan Zhang, Ziqi Yang, Han Liu, and Yang Liu. Do fine-tuned LLMs understand vulnerabilities? an investigation into the semantic trap, 2026.

[60] Felix Weissberg, Lukas Pirch, Erik Imgrund, Jonas Möller, Thorsten Eisenhofer, and Konrad Rieck. LLM-based vulnerability discovery through the lens of code metrics. In *Proc. 48th IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 2019–2031, 2026. doi: 10.1145/3744916.3764574.

[61] Youpeng Li, Weiliang Qi, Xuyu Wang, Fuxun Yu, and Xinda Wang. Revisiting pre-trained language models for vulnerability detection. In *Proc. 21st ACM ASIA Conference on Computer and Communications Security (AsiaCCS)*, pages 1540–1554, 2026. doi: 10.1145/3779208.3785291. arXiv:2507.16887.

[62] Xueying Du, Geng Zheng, Kaixin Wang, Yi Zou, Yujia Wang, Wentai Deng, Jiayi Feng, Mingwei Liu, Bihuan Chen, Xin Peng, Tao Ma, and Yiling Lou. Vul-RAG: Enhancing LLM-based vulnerability detection via knowledge-level RAG, 2024. URL `https://arxiv.org/abs/2406.11147`. Also ACM TOSEM 35(10):1–25, 2026, doi:10.1145/3797277 (introduces the PairVul vulnerable/patched benchmark).

[63] Sabrina Kaniewski, Fabian Schmidt, and Tobias Heer. Revisiting Vul-RAG: Reproducibility and replicability of RAG-based vulnerability detection with open-weight models, 2026. URL `https://arxiv.org/abs/2606.04739`. AI&CCPS 2026 workshop @ ARES 2026 (per arXiv).

[64] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Wei Ma, Lyuye Zhang, Yang Liu, and Yingjiu Li. LLM4Vuln: A unified evaluation framework for decoupling and enhancing LLMs' vulnerability reasoning, 2024. URL `https://arxiv.org/abs/2401.16185`. Technical report, NTU (v4, Jun 2025).

[65] Niklas Risse and Marcel Böhme. Uncovering the limits of machine learning for automatic vulnerability detection. In *Proc. 33rd USENIX Security Symposium*, pages 4247–4264, 2024. arXiv:2306.17193.

[66] Alperen Yildiz, Sin G. Teo, Yiling Lou, Yebo Feng, Chong Wang, and Dinil Mon Divakaran. Benchmarking LLMs and LLM-based agents in practical vulnerability detection for code repositories. In *Proc. 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 30848–30865, 2025. doi: 10.18653/v1/2025.acl-long.1490.

[67] Arastoo Zibaeirad and Marco Vieira. VulnLLMEval: A framework for evaluating large language models in software vulnerability detection and patching, 2024.

[68] Niklas Risse, Jing Liu, and Marcel Böhme. Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):388–410, 2025. doi: 10.1145/3728887.

[69] Partha Chakraborty, Krishna Kanth Arumugam, Mahmoud Alfadel, Meiyappan Nagappan, and Shane McIntosh. Revisiting the performance of deep learning-based vulnerability detection on realistic datasets. *IEEE Transactions on Software Engineering*, 50(8):2163–2177, 2024. doi: 10.1109/TSE.2024.3423712.

[70] Yansong Li, Paula Branco, Alexander M. Hoole, Manish Marwah, Hari Manassery Koduvely, Guy-Vincent Jourdan, and Stephan Jou. SV-TrustEval-C: Evaluating structure and semantic reasoning in large language models for source code vulnerability analysis. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3014–3032, 2025. doi: 10.1109/SP61157.2025.00191.

[71] Roland Croft, M. Ali Babar, and Mehdi Kholoosi. Data quality for software vulnerability datasets. In *Proc. 45th International Conference on Software Engineering (ICSE)*, pages 121–133, 2023. doi: 10.1109/ICSE48619.2023.00022.

[72] Yikun Li, Ting Zhang, Ratnadira Widyasari, Yan Naing Tun, Huu Hung Nguyen, Tan Bui, Ivana Clairine Irsan, Yiran Cheng, Xiang Lan, Han Wei Ang, Frank Liauw, Martin Weyssow, Hong Jin Kang, Eng Lieh Ouh, Lwin Khin Shar, and David Lo. CleanVul: Automatic function-level vulnerability detection in code commits using LLM heuristics, 2024.

[73] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David Wagner. DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 654–668, 2023. doi: 10.1145/3607199.3607242. URL `https://arxiv.org/abs/2304.00409`.

[74] Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David Wagner, Baishakhi Ray, and Yizheng Chen. Vulnerability detection with code language models: How far are we? In *IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 1729–1741, 2025. doi: 10.1109/ICSE55347.2025.00038. URL `https://arxiv.org/abs/2403.18624`. PrimeVul.

[75] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, 48(9):3280–3296, 2022. doi: 10.1109/TSE.2021.3087402. ReVeal.

[76] Md Basim Uddin Ahmed, Nima Shiri Harzevili, Jiho Shin, Hung Viet Pham, and Song Wang. SecVulEval: Benchmarking LLMs for real-world C/C++ vulnerability detection, 2025. URL `https://arxiv.org/abs/2505.19828`.

[77] Richard A. Dubniczky, Krisztofer Zoltán Horvát, Tamás Bisztray, Mohamed Amine Ferrag, Lucas C. Cordeiro, and Norbert Tihanyi. CASTLE: Benchmarking dataset for static code analyzers and LLMs towards CWE detection, 2025. URL `https://arxiv.org/abs/2503.09433`.

[78] Avishree Khare, Saikat Dutta, Ziyang Li, Alaia Solko-Breslin, Rajeev Alur, and Mayur Naik. Understanding the effectiveness of large language models in detecting security vulnerabilities. In *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 103–114, 2025. doi: 10.1109/ICST62969.2025.10988968.

[79] Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. LLMs cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. In *IEEE Symposium on Security and Privacy (S&P)*, pages 862–880, 2024. doi: 10.1109/SP54263.2024.00210. URL `https://arxiv.org/abs/2312.12575`.

[80] Benjamin Steenhoek, Md Mahbubur Rahman, Monoshi Kumar Roy, Mirza Sanjida Alam, Hengbo Tong, Swarna Das, Earl T. Barr, and Wei Le. To err is machine: Vulnerability detection challenges LLM reasoning, 2024. URL `https://arxiv.org/abs/2403.17218`.

[81] Xiang Mei, Jordi Del Castillo, Pulkit Singh Singaria, Haoran Xi, Abdelouahab Benchikh, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, Hammond Pearce, and Brendan Dolan-Gavitt. ARVO: Atlas of reproducible vulnerabilities for open-source software. In *2026 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 860–873, 2026. doi: 10.1109/EuroSP68448.2026.00060.

[82] Seulbae Kim, Seunghoon Woo, Heejo Lee, and Hakjoo Oh. VUDDY: A scalable approach for vulnerable code clone discovery. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 595–614, 2017. doi: 10.1109/SP.2017.62.

[83] Yiheng Cao, Susheng Wu, Ruisi Wang, Bihuan Chen, Yiheng Huang, Chenhao Lu, Zhuotong Zhou, and Xin Peng. Recurring vulnerability detection: How far are we? *Proc. ACM Softw. Eng.*, 2(ISSTA):573–595, 2025. doi: 10.1145/3728901.

[84] Seunghoon Woo, Eunjin Choi, and Heejo Lee. A large-scale analysis of the effectiveness of publicly reported security patches. *Computers & Security*, 148:104181, 2025. doi: 10.1016/j.cose.2024.104181.

[85] Weiliang Qi, Youpeng Li, and Xinda Wang. Why not fix it once and for all? an empirical study of multiple patches for vulnerability fixes in open-source software, 2026. Accepted at ESORICS 2026.

[86] Kent Walker. A summer of security: empowering cyber defenders with AI. Google Blog (The Keyword), July 2025. URL https://blog.google/technology/safety-security/cybersecurity-updates-summer-2025/. Posted 2025-07-15; Big Sleep found SQLite CVE-2025-6965.

[87] Raluca Ada Popa and Four Flynn. Introducing CodeMender: an AI agent for code security. Google DeepMind blog, October 2025. URL https://deepmind.google/blog/introducing-codemender-an-ai-agent-for-code-security/. Posted 2025-10-06.

[88] OpenAI. Introducing Aardvark: OpenAI's agentic security researcher. OpenAI blog, October 2025. URL https://openai.com/index/introducing-aardvark/. Announced 2025-10-30 (private beta, GPT-5 based).

[89] Carl Franzen. Meet Aardvark, OpenAI's security agent for code analysis and patching. VentureBeat, October 2025. URL https://venturebeat.com/security/meet-aardvark-openais-in-house-security-agent-for-code-analysis-and-patching. Published 2025-10-30. "The system integrates with OpenAI Codex to generate patches.".

[90] DARPA. AI Cyber Challenge marks pivotal inflection point for cyber defense. DARPA news release, August 2025. URL https://www.darpa.mil/news/2025/aixcc-results. Posted 2025-08-08; winners Team Atlanta, Trail of Bits, Theori.

[91] Taesoo Kim et al. ATLANTIS: AI-driven threat localization, analysis, and triage intelligence system, 2025. URL https://arxiv.org/abs/2509.14589. Team Atlanta, AIxCC 1st place; 46 authors.

[92] Trail of Bits. Buttercup, an open-source AI cyber reasoning system, 2025. URL https://www.trailofbits.com/buttercup/. Open-sourced 2025-08-08.

[93] Ze Sheng, Qingxiao Xu, Jianwei Huang, Matthew Woodcock, Heqing Huang, Alastair F. Donaldson, Guofei Gu, and Jeff Huang. All you need is a fuzzing brain: An LLM-powered system for automated vulnerability detection and patching, 2025.

[94] Cen Zhang, Younggi Park, Fabian Fleischer, Yu-Fu Fu, Jiho Kim, Dongkwan Kim, Youngjoon Kim, Qingxiao Xu, Andrew Chin, Ze Sheng, Hanqing Zhao, Michael Pelican, David J. Musliner, Jeff Huang, Jon Silliman, Mikel Mcdaniel, Jefferson Casavant, Isaac Goldthwaite, Nicholas Vidovich, Matthew Lehman, and Taesoo Kim. SoK: DARPA's AI Cyber Challenge (AIxCC): Competition design, architectures, and lessons learned. In *USENIX Security Symposium*, pages 2705–2724, 2026. URL https://arxiv.org/abs/2602.07666.

[95] Andrew Chin, Dongkwan Kim, Yu-Fu Fu, Fabian Fleischer, Youngjoon Kim, HyungSeok Han, Cen Zhang, Brian Junekyu Lee, Hanqing Zhao, and Taesoo Kim. OSS-CRS: Liberating AIxCC cyber reasoning systems for real-world open-source security. In *Proc. 20th USENIX WOOT Conference on Offensive Technologies (WOOT '26)*, pages 305–320, 2026.

[96] Ze Sheng, Zhicheng Chen, Qingxiao Xu, Kewen Zhu, and Jeff Huang. FuzzingBrain V2: A multi-agent LLM system for automated vulnerability discovery and reproduction, 2026.

[97] Hanzhi Liu, Chaofan Shou, Xiaonan Liu, Hongbo Wen, Yanju Chen, Ryan Jingyang Fang, and Yu Feng. Synthesizing multi-agent harnesses for vulnerability discovery, 2026.

[98] Yunlong Lyu, Yuxuan Xie, Peng Chen, and Hao Chen. Prompt fuzzing for fuzz driver generation. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 3793–3807, 2024. doi: 10.1145/3658644.3670396. arXiv:2312.17677.

[99] Cen Zhang, Yaowen Zheng, Mingqiang Bai, Yeting Li, Wei Ma, Xiaofei Xie, Yuekang Li, Limin Sun, and Yang Liu. How effective are they? exploring large language model based fuzz driver generation. In *Proc. 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 1223–1235, 2024. doi: 10.1145/3650212.3680355.

[100] Google. OSS-Fuzz-Gen: LLM powered fuzzing via OSS-Fuzz, 2024. URL https://github.com/google/oss-fuzz-gen.

[101] Dongge Liu, Jonathan Metzman, and Oliver Chang. AI-powered fuzzing: Breaking the bug hunting barrier. Google Online Security Blog, August 2023. URL https://security.googleblog.com/2023/08/ai-powered-fuzzing-breaking-bug-hunting.html. Posted 2023-08-16.

[102] Oliver Chang, Dongge Liu, and Jonathan Metzman. Leveling up fuzzing: Finding more vulnerabilities with AI. Google Online Security Blog, November 2024. URL https://security.googleblog.com/2024/11/leveling-up-fuzzing-finding-more.html. Posted 2024-11-20; 26 vulns incl. OpenSSL CVE-2024-9143.

[103] Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. CyberGym: Evaluating AI agents' real-world cybersecurity capabilities at scale, 2025. URL https://arxiv.org/abs/2506.02548.

[104] Hwiwon Lee, Ziqi Zhang, Hanxiao Lu, and Lingming Zhang. SEC-bench: Automated benchmarking of LLM agents on real-world software security tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 128940–128976, 2025.

[105] Ze Sheng, Aleksandar Kezic, Zhicheng Chen, and Jeff Huang. FuzzingBrain-Bench V1: Evaluating open-ended bug discovery by LLMs, 2026.

[106] Anthropic Frontier Red Team. Building AI for cyber defenders. Anthropic research blog, September 2025. URL https://www.anthropic.com/research/building-ai-cyber-defenders. Posted 2025-09-29 (Claude Sonnet 4.5; CyberGym/Cybench results).

[107] Stanislav Fort. AI cybersecurity after Mythos: The jagged frontier. AISLE blog, April 2026. URL https://aisle.com/blog/ai-cybersecurity-after-mythos-the-jagged-frontier. Posted 2026-04-07, updated 2026-04-09. Code/transcripts: https://github.com/stanislavfort/mythos-jagged-frontier.

[108] Stanislav Fort. System over model: Zero-day discovery at the jagged frontier. AISLE blog, April 2026. URL https://aisle.com/blog/system-over-model-zero-day-discovery-at-the-jagged-frontier. Posted 2026-04-14 (nano-analyzer).

[109] Kurt Boberg. Needles and haystacks: Can open-source & flagship models do what Mythos did? Semgrep blog, https://semgrep.dev/blog/2026/needles-and-haystacks-can-open-source-flagship-models-do-what-mythos-did/, 2026. 2026-04-17.

[110] clearbluejar. System over model, tested: Reproducing Mythos's FreeBSD find on local open-weight models. https://clearbluejar.github.io/posts/system-over-model-tested-mythos-freebsd-local-openweight/, 2026. 2026-06-04.

[111] Dawid Moczadło, Klaudia Kloc, Marek Lewandowski, Amadeusz Lisiecki, Jakub Sienkiewicz, and Mikołaj Palkiewicz. We reproduced Anthropic's Mythos findings with public models. Vidoc Security Lab blog, April 2026. URL https://blog.vidocsecurity.com/blog/we-reproduced-anthropics-mythos-findings-with-public-models. Posted 2026-04-14.

[112] Isaac David and Arthur Gervais. Benchmarking Mythos-linked bug rediscovery, 2026. URL https://arxiv.org/abs/2605.17416.

[113] Zhengxiong Luo, Mehtab Zafar, Dylan Wolff, and Abhik Roychoudhury. Code-Augur: Agentic vulnerability detection via specification inference, 2026. URL https://arxiv.org/abs/2606.18619.

[114] Sean Heelan. How I used o3 to find CVE-2025-37899, a remote zeroday vulnerability in the Linux kernel's SMB implementation. https://sean.heelan.io/2025/05/22/how-i-used-o3-to-find-cve-2025-37899-a-remote-zeroday-vulnerability-in-the-linux-kernels-smb-implementation/, 2025.

[115] AISLE Research Team. AISLE discovered 12 out of 12 OpenSSL vulnerabilities. https://aisle.com/blog/aisle-discovered-12-out-of-12-openssl-vulnerabilities, 2026. vendor report, 2026-01-26.

[116] Daniel Stenberg. A new breed of analyzers. https://daniel.haxx.se/blog/2025/10/10/a-new-breed-of-analyzers/, 2025.

[117] Alfredo Pesoli, Herman Errico, and Lorenzo Cavallaro. Demystifying the Mythos or disrupting bugonomics? from zero-day asymmetry to defender remediation throughput, 2026. URL https://arxiv.org/abs/2605.24632.

[118] CISA and NSA and FBI and international partners. The case for memory safe roadmaps, December 2023. URL https://www.cisa.gov/resources-tools/resources/case-memory-safe-roadmaps. Published 2023-12-06.

[119] LLVM Project. AddressSanitizer – Clang documentation, 2026. URL https://clang.llvm.org/docs/AddressSanitizer.html.

[120] GNU C Library. Source fortification (The GNU C Library manual, D.2), 2026. URL https://sourceware.org/glibc/manual/latest/html_node/Source-Fortification.html. _FORTIFY_SOURCE levels 1-3.

[121] GCC Project. Program instrumentation options (Using the GNU Compiler Collection), 2026. URL https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html. -fstack-protector(-strong), -fsanitize=address.

[122] Michal Zalewski. american fuzzy lop (AFL): a security-oriented fuzzer, 2013. URL https://github.com/google/AFL. repo states "Originally developed by Michal Zalewski".

[123] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. AFL++: Combining incremental steps of fuzzing research. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020. URL https://www.usenix.org/conference/woot20/presentation/fioraldi.

[124] LLVM Project. libFuzzer – a library for coverage-guided fuzz testing, 2026. URL https://llvm.org/docs/LibFuzzer.html.

[125] Google. OSS-Fuzz: Documentation, 2026. URL https://google.github.io/oss-fuzz/.

[126] Repology. Repology: the packaging hub. https://repology.org/, 2026.

[127] IBM Granite Team. granite-guardian-3.1-2b. Hugging Face model card, https://huggingface.co/ibm-granite/granite-guardian-3.1-2b, 2024. 2.53B parameters, Apache-2.0, published 2024-12-17.

[128] IBM Granite Team. granite-guardian-3.2-5b. Hugging Face model card, https://huggingface.co/ibm-granite/granite-guardian-3.2-5b, 2025. 5.78B parameters, Apache-2.0, published 2025-01-23.

[129] IBM Granite Team. granite-guardian-3.3-8b. Hugging Face model card, https://huggingface.co/ibm-granite/granite-guardian-3.3-8b, 2025. 8.17B parameters, Apache-2.0, published 2025-06-03.

[130] Microsoft. Phi-4-mini-instruct. Hugging Face model card, https://huggingface.co/microsoft/Phi-4-mini-instruct, 2025. 3.84B parameters, MIT license, published 2025-02-19.

[131] Salesforce AI Research. xLAM-2 function-calling models: xlam-2-1b-fc-r and xlam-2-3b-fc-r. Hugging Face model cards, https://huggingface.co/Salesforce/xLAM-2-1b-fc-r, https://huggingface.co/Salesforce/xLAM-2-3b-fc-r, 2025. 1.54B and 3.09B parameters, CC-BY-NC-4.0, published 2025-03-27.

[132] FIRST. Common vulnerability scoring system v3.1: Specification document. Forum of Incident Response and Security Teams, n.d. URL https://www.first.org/cvss/v3.1/specification-document. Accessed 2026-11-02. Added from co-author review (Gkogkou, 2026-11-02).

[133] TechnoEdge editorial staff. NVIDIA DGX Spark price raised 1.5× to US$6,950; 64 GB model added at US$4,999 (in Japanese). TechnoEdge, October 2026. URL https://www.techno-edge.net/article/2026/10/08/5565.html. Published 2026-10-08. 128 GB model: US$3,999 at launch (October 2025), US$4,699 from February 2026, US$6,950 (shown sold out) in October 2026. Title translated from Japanese.

[134] NVIDIA Developer Forums. 2/23/2026 price change announcement (DGX Spark). https://forums.developer.nvidia.com/t/2-23-2026-price-change-announcement/361713, 2026. DGX Spark Founders Edition MSRP raised from US$3,999 to US$4,699, posted 2026-02-25, all regions, citing memory supply constraints.

[135] Google. diffusiongemma-26b-a4b-it. Hugging Face model card, https://huggingface.co/google/diffusiongemma-26B-A4B-it, 2026. Card: 25.2B total / 3.8B active MoE (25.8B safetensors incl. vision encoder), Apache-2.0, published 2026-06-09.

[136] NVIDIA. diffusiongemma-26b-a4b-it-nvfp4. Hugging Face model card, https://huggingface.co/nvidia/diffusiongemma-26B-A4B-it-NVFP4, 2026. NVFP4 quantisation of google/diffusiongemma-26B-A4B-it; card lists 25.2B total and 3.8B active parameters; Apache-2.0 with Gemma terms; published 2026-06-10.

A Known-CVE recall set

Selection rule. The selection script fixed the rule and random seed before its single run. The written evaluation plan specified the dates, CWE set, C/C++ fix, two-family packaging, status-sheet exclusion, 1–3-function limit and a seeded draw of 75. The script added one CVE per repository, the 2 GB size limit and the seed value. We used only public sources.

1. NVD CVEs published from 2023-01-01 to 2026-09-30 whose CWE is one of 787, 121, 122, 125, 416, 190, 191, 131 or 119, and which reference a fix commit on GitHub.
2. The fix changes C or C++ files and 1–3 functions (counted from the diff's hunk headers).
3. The project is packaged in at least two Repology families, and its name does not match a project on the campaign's disclosure status sheet (case- and punctuation-insensitive; names of five or more characters also match as substrings).
4. One CVE per repository; repository size at most 2 GB.
5. Candidates are shuffled with seed 20261028 and taken in order until 75 are accepted.

Of 519 CVEs with a GitHub fix commit, 290 were considered in seeded order to accept 75. The main reasons for rejection were: a repository already selected, fewer than two Repology families, no C/C++ change, a project on the status sheet, or a fix spanning 0 or more than 3 functions. The accepted set is 41 CWE-787, 19 CWE-125, 8 CWE-416, 5 CWE-190, 1 CWE-119 and 1 CWE-121.

Targets and counting. Each repository is checked out at the parent of the fix commit. Target functions are those whose code at that commit overlaps a line changed by the fix. Windows of one large function count as that function. A CVE is found at k if any target function is among the repository's k highest-scored functions, with ties counted against us. Because hunk headers can name a neighbouring function, we report the line-based targets. Counting by hunk-header names gives 10, 16 and 17 at 10, 25 and 40. Counting only the exact window containing the change gives 8, 11 and 12.

Table 8: The 75 public CVEs of the recall evaluation, with the fix commit used (the repository is checked out at its parent). Repositories are as referenced by NVD; one reference points to a GitHub mirror of the project.

CVE	CWE	Repository	Fix commit
CVE-2021-33796	416	ccxvii/mujs	`7ef066a3bb95`
CVE-2021-45985	787	lua/lua	`cf613cdc6fa3`
CVE-2022-28071	416	radareorg/radare2	`65448811e5b9`
CVE-2022-48468	190	protobuf-c/protobuf-c	`ec3d900001a1`
CVE-2023-0341	787	editorconfig/editorconfig-core-c	`41281ea82fbf`
CVE-2023-2804	787	libjpeg-turbo/libjpeg-turbo	`9f756bc67a84`
CVE-2023-4255	787	tats/w3m	`edc602651c50`
CVE-2023-4755	416	gpac/gpac	`895ac12da168`
CVE-2023-5217	787	webmproject/libvpx	`3fbd1dca6a4d`
CVE-2023-5341	416	imagemagick/imagemagick	`aa673b2e4def`
CVE-2023-23457	119	upx/upx	`779b648c5f6a`
CVE-2023-25567	125	gssapi/gss-ntlmssp	`025fbb756d44`
CVE-2023-29418	125	kspalaiologos/bzip3	`aae16d107f80`
CVE-2023-34095	787	openprinting/cpdb-libs	`f181bd1f1475`
CVE-2023-36274	787	libredwg/libredwg	`8651fa27dd2d`
CVE-2023-41910	125	lldpd/lldpd	`a9aeabdf879c`
CVE-2023-47470	787	ffmpeg/ffmpeg	`4565747056a1`
CVE-2023-49285	125	squid-cache/squid	`77b3fb4df0f1`
CVE-2023-50268	787	jqlang/jq	`c9a51565214e`
CVE-2024-8946	787	micropython/micropython	`29943546343c`
CVE-2024-13903	787	quickjs-ng/quickjs	`99c02eb45170`
CVE-2024-25629	125	c-ares/c-ares	`a804c04ddc82`
CVE-2024-32658	125	freerdp/freerdp	`1a755d898ddc`
CVE-2024-37407	125	libarchive/libarchive	`b6a979481b7d`
CVE-2024-45306	787	vim/vim	`396fd1ec2956`
CVE-2025-9300	787	saitoha/libsixel	`316c086e79d6`
CVE-2025-9394	416	podofo/podofo	`22d16cb142f2`
CVE-2025-12875	787	makesoftwaresafe/mruby	`93619f06dd37`
CVE-2025-13654	787	zevv/duc	`8638c4365ffd`
CVE-2025-14956	125	webassembly/binaryen	`4f52bff8c407`
CVE-2025-14958	787	seyhajin/sokol	`33e2271c431b`
CVE-2025-15155	787	floooh/sokol	`5d1134415097`
CVE-2025-15555	787	open5gs/open5gs	`54dda0412110`
CVE-2025-34451	787	httpsgithu/proxychains-ng	`cc005b713281`
CVE-2025-43962	125	libraw/libraw	`66fe663e02a4`
CVE-2025-58050	787	pcre2project/pcre2	`a141712e5967`

CVE	CWE	Repository	Fix commit
CVE-2025-64330	787	oisf/suricata	`482e5eac9218`
CVE-2025-64524	787	openprinting/cups-filters	`956283c74a34`
CVE-2025-68973	787	gpg/gnupg	`115d138ba599`
CVE-2026-1418	787	enocknt/gpac	`10c73b82cf0e`
CVE-2026-1998	787	dpgeorge/micropython	`570744d06c5b`
CVE-2026-2016	787	happyfish100/libfastcommon	`82f66af3e252`
CVE-2026-2258	787	aardappel/lobster	`c2047a33e1ac`
CVE-2026-3308	190	artifexsoftware/mupdf	`a26f0142e7d3`
CVE-2026-3945	190	tinyproxy/tinyproxy	`969852ccdb1d`
CVE-2026-12630	125	zephyrproject-rtos/zephyr	`1bbb7aefa69e`
CVE-2026-14461	125	traviscross/mtr	`48e1794414d3`
CVE-2026-18238	125	the-tcpdump-group/libpcap	`b9590d482986`
CVE-2026-18370	787	eradman/entr	`2467fe0aa5f2`
CVE-2026-38821	787	opennds/opennds	`3b5f7ef40cd0`
CVE-2026-39919	787	artifexsoftware/ghostpdl	`0a8bf88e39db`
CVE-2026-44235	125	alanxz/rabbitmq-c	`1d3afbb056fe`
CVE-2026-45809	787	opensips/opensips	`c5970d3ee25b`
CVE-2026-48977	787	openslide/openslide	`2be88bd782d9`
CVE-2026-50144	787	tencent/ncnn	`5a0288f255da`
CVE-2026-50538	787	libvnc/libvncserver	`540332be3e0a`
CVE-2026-53587	125	libgit2/libgit2	`2c0ce8c0132a`
CVE-2026-53938	787	openidc/cjose	`8c51d2452735`
CVE-2026-54616	125	m2team/nanazip	`733e8570d2ba`
CVE-2026-55777	125	allinurl/goaccess	`ba813ed97d99`
CVE-2026-56109	787	alsa-project/alsa-lib	`536dd6f8affd`
CVE-2026-56684	416	valkey-io/valkey	`7cd5bcb7575d`
CVE-2026-57162	121	pjsip/pjproject	`a1b707c0c9b0`
CVE-2026-61548	787	rsyslog/rsyslog	`bcda60a3692e`
CVE-2026-61714	125	fluidsynth/fluidsynth	`772702e00cc6`
CVE-2026-62959	125	coturn/coturn	`960835886692`
CVE-2026-63388	787	libevent/libevent	`52057cb33d0c`
CVE-2026-65832	125	deskflow/deskflow	`205a3c803e52`
CVE-2026-68768	787	hashcat/hashcat	`68f56a2d8712`
CVE-2026-76844	787	madler/zlib	`df84af25dc19`
CVE-2026-77219	190	emacs-mirror/emacs	`b07e634e4cf4`
CVE-2026-77358	416	yhirose/cpp-httplib	`2f986fd5e56e`
CVE-2026-81934	416	redis/redis	`6d088c335d5c`
CVE-2026-87823	190	luben/zstd-jni	`46dcb2a54c8b`
CVE-2026-102296	787	zoneminder/zoneminder	`2596e5fb64c0`