

---

# FROM TRACE TO LINE: LLM AGENT FOR REAL-WORLD OSS VULNERABILITY LOCALIZATION

## ABSTRACT

Large language models show promise for vulnerability discovery, yet prevailing methods inspect code in isolation, struggle with long contexts, and focus on coarse function or file level detections which offers limited actionable guidance to engineers who need precise line-level localization and targeted patches in real-world software development. We present T2L-Agent (Trace-to-Line Agent), a project-level, end-to-end framework that plans its own analysis and progressively narrows scope from modules to exact vulnerable lines. T2L-Agent couples multi-round feedback with an Agentic Trace Analyzer (ATA) that fuses runtime evidence such as crash points, stack traces, and coverage deltas with AST-based code chunking, enabling iterative refinement beyond single pass predictions and translating symptoms into actionable, line-level diagnoses. To benchmark line-level vulnerability discovery, we introduce T2L-ARVO, a diverse, expert-verified 50-case benchmark spanning five crash families and real-world projects. T2L-ARVO is specifically designed to support both coarse-grained detection and fine-grained localization, enabling rigorous evaluation of systems that aim to move beyond file-level predictions. On T2L-ARVO, T2L-Agent achieves up to 58.0% detection and 54.8% line-level localization, substantially outperforming baselines. Together, the framework and benchmark push LLM-based vulnerability detection from coarse identification toward deployable, robust, precision diagnostics that reduce noise and accelerate patching in open-source software workflows.

## 1 INTRODUCTION

Software vulnerabilities now occur at unprecedented scale and cost. In 2023, more than 29,000 Common Vulnerabilities and Exposures (CVEs) were recorded CVE Details (2024). In the first half of 2025, 1,732 data breaches were reported—an 11% increase Indusface (2024). The economic toll is mounting: software supply-chain attacks are projected to cost the global economy 80.6 billion annually by 2026, up from 45.8 billion in 2023 Dark Reading (2024); some estimates put total damages at \$9.5 trillion in 2024 Liu et al. (2024). Critically, 14% of breaches in 2024 began with vulnerability exploitation—nearly triple the prior year. Despite advances in automated vulnerability detection, effectiveness in localization remains underexplored Zhang et al. (2024b), leaving developers with coarse, file-level predictions while 32% of critical vulnerabilities remain unpatched for over 180 days. These vulnerabilities are exploding in volume and growing ever more severe, driving escalating security and economic risks worldwide.

**Motivation.** The rise of Large Language Models (LLMs) has accelerated AI-driven automation across software development and cybersecurity Zhang et al. (2025a). From code assistants like GitHub, Inc. (2021) to automated security analysis and automation Nunez et al. (2024). LLMs show strong aptitude for understanding complex codebases and flagging potential issues Divakaran & Peddinti (2024). This momentum naturally extends to vulnerability detection, spanning model-level approaches such as fine-tuning LLMs for classification—and agentic frameworks that harness LLM reasoning for automated analysis tmylla and contributors (2024).

Yet today’s LLM-based detection methods face practical barriers. Most operate at the function level, asking whether a fragment is vulnerable rather than pinpointing where the flaw lies Zhang et al. (2024b), Sovrano et al. (2025). These tasks are also defined purely on static code, ignoring binaries, runtime behavior, and project-level diffs, further limiting their usefulness for real remedi-

---

ation. Evaluations also rely heavily on lightweight, synthetic datasets that miss the complexity of production systems Guo et al. (2024).

This research–practice gap is most visible in vulnerability localization. While studies show promising results on isolated functions or snippets, engineers must navigate large repositories with cross-module dependencies and need line-level localization to craft minimal, targeted patches precision current methods rarely deliver. Moreover, artificial benchmarks obscure the true difficulty of project-level detection, where context spans multiple files and demands system-wide reasoning.

Given the surge in vulnerability reports outlined earlier, the field urgently needs approaches that close this reality gap: leveraging LLM automation while tackling project-scale challenges practitioners face daily. Only by embracing these real-world constraints can we convert the promise of LLM-assisted security into tools that meaningfully lighten the load on development teams.

To address these gaps, we introduce T2L (Trace-to-Line), which reframes vulnerability detection as a two-tier problem (a) coarse-grained detection - flagging suspicious code chunks and (b) fine-grained localization - pinpointing exact vulnerable lines. This separation enables systematic evaluation of LLM capabilities from repository-scale reasoning to human-expert precision.

**Contribution.** This work makes three contributions: **(1) *New Task formulation.*** We define a new task for agentic AI, a runtime trace guided, project-level vulnerability detection, and formulate it as two structured subtasks: chunk-level detection and line-level localization. AST-based chunking adapts large codebases to LLM context while preserving semantics, enabling refinement from coarse predictions to exact lines, bridging research setups and real-world needs for LLMs. **(2) *T2L-ARVO benchmark.*** We present the first benchmark for agentic fine-grained localization, featuring 50 expert verified cases across five vulnerability types with balanced category distribution. T2L-ARVO enables realistic, project-scale evaluation of LLM-based systems. **(3) *T2L-Agent framework.*** We propose a multi-agent system with the following innovation: (i) a Trace Analyzer (Agentic Trace Analyzer) integrating the ensemble of tools such as static analysis, sanitizers, and runtime monitoring for observability as human debuggers; (ii) a Proposal module (Divergence Tracing) that iteratively forms and tests vulnerability hypotheses; and (iii) a two-stage pipeline (Detection Refinement) that first detects coarse chunks and then verifies exact lines with runtime feedback.

Overall, T2L-Agent provides a comprehensive baseline for this new vulnerability localization task. Unlike prior snippet-level detectors, it is designed to operate on fully reproducible real-world projects, using crashes, sanitizer reports, and project diffs as runtime evidence. On T2L-ARVO, it achieves up to **58%** chunk-level detection and **54.8%** exact line localization, showing that this runtime-evidence–guided setting is challenging yet tractable for LLM agents.

## 2 BACKGROUND

**Vulnerability Localization.** Classical localization combines static and dynamic analyses with retrieval and graph methods. Static slicing selects statements that may affect a slicing criterion to form behavior-preserving slices for debugging and comprehension Weiser (1981). Dynamic slicing refines this by deriving input-specific slices from execution-time dependence graphs, yielding higher precision near crashes or variables Agrawal & Horgan (1990). Information-retrieval approaches treat bug reports as queries, ranking files/methods by textual and historical signals to surface likely locations Zhou et al. (2012); Saha et al. (2013); Wang & Lo (2014). Code Property Graphs unify classic program-analysis concepts in a joint representation, enabling scalable pattern searches that uncovered 18 previously unknown vulnerabilities Yamaguchi et al. (2014).

**AI for Cybersecurity.** AI is widely applied to cybersecurity tasks with tool execution and feedback in runnable environments Yang et al. (2023). One prominent use is offensive security Shao et al. (2024b), where agents reason about attack chains and interact with benchmark environments Shao et al. (2024c); Zhang et al. (2024a) to simulate human CTF players Udeshi et al. (2025); Shao et al. (2025); Abramovich et al. (2025). In parallel, pen-testing systems orchestrate modules or multi-agent collaboration to automate reconnaissance, vulnerability analysis, and exploitation against targets Deng et al. (2024); Shen et al. (2025). Further work builds agent-based patching frameworks that localize vulnerabilities and synthesize fixes Yu et al. (2025); Xue et al. (2025), and explores LLM-augmented red and blue teaming, where agents emulate adaptive attackers and assist defenders in threat hunting and incident investigation Abuadbbba et al. (2025); Liu et al. (2025).

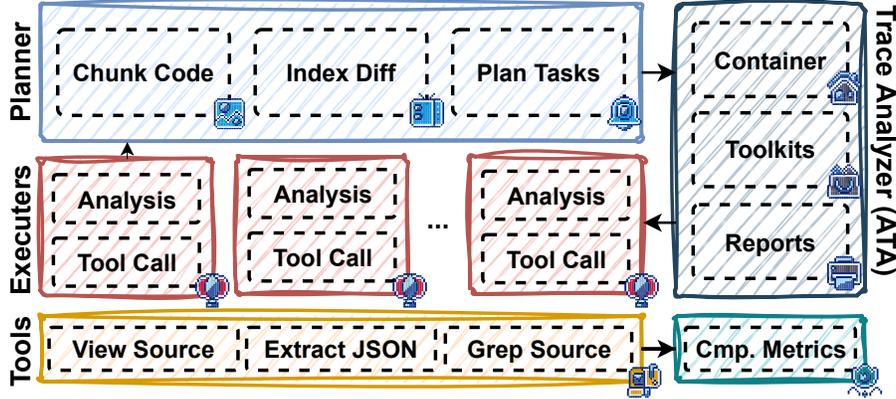


Figure 1: T2L-Agent Framework overview.

**LLM Agentic Systems.** Beyond cybersecurity, large language models (LLMs) are increasingly applied to automate workflows across diverse domains Shao et al. (2024a), including AI for scientific research Bran et al. (2023); Basit et al. (2025); Jin et al. (2024), software engineering Yang et al. (2024c), healthcare Isern & Moreno (2016), and hardware design Wang et al. (2024). Many systems integrate long-term memory to retain intermediate results, environment states, and user preferences across sessions Zhang et al. (2025b). LLM as judge are often used to evaluate actions, select among tool outputs, and provide feedback for self-improvement Li et al. (2025). Some agents employ self-reflection and critique loops to iteratively refine their reasoning or code before execution Renze & Guven (2024). Multi-agent and role-specialized architectures further enable collaboration among agents with distinct skills to solve complex, interleaved subtasks in parallel Li et al. (2024a).

### 3 RELATED WORK

LLMs show strong potential on vulnerability localization tasks. Early learning-based detectors improve localization by training models to classify whether a unit is vulnerable. LLMAO Yang et al. (2024a) fine-tunes LLMs on small, manually curated buggy programs, while BAPStein et al. (2025) learns state-of-the-art vulnerability localization without any direct localization labels, outperforming traditional baseline over eight benchmarks. However, subsequent analyses revealed data issues in widely used datasets such as Big-Vul and Devign Zhou et al. (2019), making people doubt about performance numbers and highlighting the need for realistic evaluation settings Croft et al. (2023).

Many prior works frame vulnerability localization at the file or function level, but this coarse granularity often fails to offers limited guidance for developers. GenLoc Asad et al. (2025) identifies potentially vulnerable files from bug reports and iteratively analyzes them using code exploration tools. AgentFL Qin et al. (2024) applies a multi-agent framework for function-level localization, modeling the task as a three-step pipeline with specialized agents and tools. CoSIL Jiang et al. (2025) narrows the function-level search space using module call graphs and iteratively traverses them for relevant context. Similarly, AutoFL Kang et al. (2024) prompts LLMs to localize method-level vulnerabilities via function-call navigation, showing that multi-step reasoning mitigates context limits.

Recent studies have shifted toward line or statement level localization. LineVul Fu & Tantithamthavorn (2022) uses a Transformer-based classifier for line-level prediction, while LOVA Li et al. (2024b) introduces a self-attention framework to score and highlight vulnerable lines. MatsVD Weng et al. (2024) enhances statement level localization with dependency-aware attention, and xLoc Yang et al. (2024b) learns multilingual, task-specific knowledge for bug detection and localization. Building on these efforts, LLM4FL Rafi et al. (2024) proposes a multi-agent framework leveraging graph-based retrieval and navigation to reason about failure causes. MemFL Yeo et al. (2025) introduces external memory to incorporate project-specific knowledge, improving localization in complex, repository-scale systems. They push localization to line level and use more multi-agent coordination. However, most still rely on limited runtime evidence, single-pass predictions, or benchmarks that lack realistic project settings.

Study	Line Lv.	Mult. Ag.	Runtime	Iterative
LLMAO 2024a	✓	✗	✗	✗
BAP 2025	✓	✗	✗	✗
GenLoc 2025	✗	✗	✓	✓
AgentFL 2024	✗	✗	✓	✓
CoSIL 2025	✗	✗	✗	✓
AutoFL 2024	✗	✓	✓	✓
LineVul 2022	✓	✗	✗	✗
LOVA 2024b	✓	✗	✗	✗
MatsVD 2024	✗	✗	✗	✗
xLoc 2024b	✓	✗	✗	✗
LLM4FL 2024	✗	✗	✓	✓
MemFL 2025	✗	✗	✓	✓
<b>T2L (ours)</b>	✓	✓	✓	✓

Figure 2: Related Works

T2L formalizes a new task of runtime-trace-guided vulnerability localization at the project level. Prior learning-based approaches such as BAP, LLMAO, and LineVul rely on static code analysis, training models to classify whether a given code snippet is vulnerable based solely on syntactic and semantic patterns. These methods typically operate on isolated single files and produce coarse-grained predictions at the function or file level without leveraging dynamic execution information. In contrast, our task targets multi-file codebases and incorporates runtime evidence including reproducible crash logs, stack traces, and sanitizer reports to pinpoint exact vulnerable lines. This formulation more closely reflects real-world vulnerability triage workflows where developers rely on both static code inspection and dynamic program behavior to diagnose security flaws.

## 4 METHODS

### 4.1 T2L AGENT

T2L-Agent uses a hierarchical planner-executor design Udeshi et al. (2025) that splits localization into evidence gathering, hypothesis generation, and iterative refinement. Unlike single-pass analyzers, it uses a human-like process: gather runtime signals, link them to code, and narrow the search space. Details of the tool list that T2L originally supports are provided in A.5.

**Evidence Tracing** T2L Planner coordinates repository analysis and runtime evidence capture in a single, structured pipeline. First, code-structure analysis partitions the codebase into function-aligned, semantically coherent units that preserve syntactic relationships while fitting LLM context windows; known patch locations are also indexed to establish evaluation baselines. Next, runtime evidence becomes the cornerstone: Sanitizer records memory-violation patterns, allocation traces, and stack frames, while interactive debugging (GDB/LLDB) provides symbolic context via backtraces and variable-state snapshots at crash points. This dual-layer design yields comprehensive, observable crash logs rather than speculative static signals. We collect multiple types of runtime and static evidence, including crash logs, sanitizer reports, stack traces, and static analyzer results, all evidence is merged into a single unified textual block and fed to the LLM at once. This design ensures that the LLM forms its reasoning based on the global evidence context, rather than aggregating per-evidence predictions to avoid issues of cross-signal inconsistency. The integrated workflow is a key T2L innovation; we detail it in Sec. 4.1.

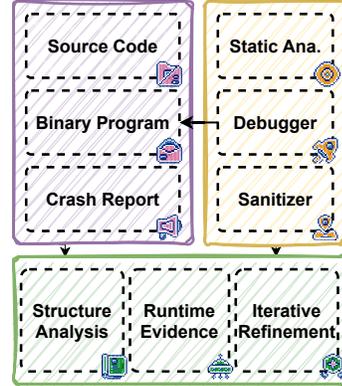


Figure 3: ATA Components

**Hypothesis Generation** A “hypothesis” an LLM-generated candidate vulnerability produced from the unified evidence block of an existing vulnerable project in T2L system. The LLM proposes a list of `file:line` candidate pairs ranked by confidence. Each hypothesis is independently checked against the ground-truth patch diff, and candidates do not affect one another during evaluation.

**Two-stage Refinement** To ensure accurate detection, T2L uses a two-stage refinement process. It begins with `coarse-grained filtering`: the system derives broad candidate regions from crash logs, using LLM code comprehension to link symptoms to likely causes and produce a ranked list of candidates with confidence and rationale. It then identifies code fragments the LLM flagged as relevant in this first pass—such as missing bounds checks, uninitialized variables, or improper memory management. Next comes `fine-grained refinement`: these extracted fragments are added to the evidence for a second LLM pass, which updates the candidate set. Any newly generated locations are merged into the existing candidates. This refinement loop continues until the LLM produces no additional candidates or the cost budget is reached.

**Feedback Control** Each cycle outputs a brief task summary with success indicators and confidence. The Planner adapts the next step—continue refining or stop, preventing premature termination and avoiding over-analysis. The loop improves precision while controlling compute cost.

**Agentic Trace Analyzer (ATA)** The Agentic Trace Analyzer (ATA) is the core module of T2L-Agent. Its purpose is to connect runtime behavior with relevant source code regions, addressing a gap in prior work that relies mainly on static signals. ATA constructs a unified “evidence block” by running targets in reproducible Docker, partitioning repositories into semantically meaningful chunks using tree-sitter, and instrumenting executions with analysis toolkit such as Sanitizers,

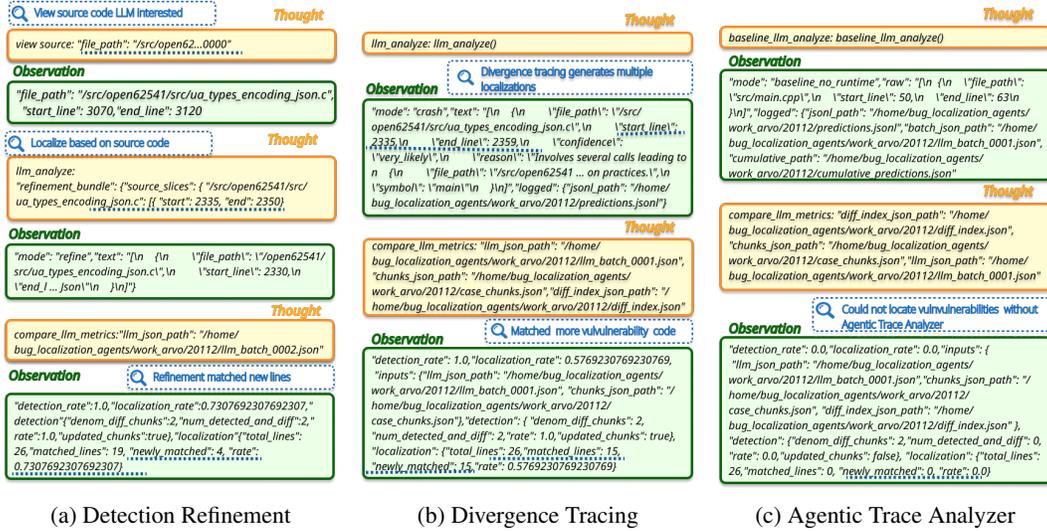


Figure 4: Partial T2L-Agent logs to show the how the three proposed technique on T2L-Agent work and help the task: Detection Refinement, Divergence Tracing and Agentic Trace Analyzer.

Debuggers, and Static analyzers to collect stack traces, memory errors, and other runtime cues. The output is a unified “evidence block” which is then passed to T2L-Agent’s reasoning modules.

We use hierarchical refinement, starting from crash logs and narrowing to specific code. The agent cross-validates runtime signals with static features and ranks candidates by LLM confidence. Like human debugging, T2L-Agent seeds candidates via static–dynamic correlation and iteratively refines them with source–feedback loops. ATA brings fewer single-shot failures, improved compute efficiency, and behavior-anchored decisions that enable precise, line-level localization even in large, tightly coupled codebases. With ATA disabled, the LLM localizes on its own, as shown in Figure 4(c). Without ATA, the LLM fails to localize vulnerabilities compared with (a) and (b).

## 4.2 T2L FEATURES

**Divergence Tracing.** Recognizing that complex vulnerabilities often involves multiple files and functions, T2L-Agent also uses divergence tracing to explore multiple hypotheses in parallel from the same crash log. This feature is inspired by how modern LLM interfaces offer multiple response variations for users, using the LLM’s variability to ensure comprehensive coverage of potential vulnerability locations. Rather than committing to a single chain of thought, it expands several in parallel and returns a ranked list of candidate sites across the search space. This surfaces correct localizations that were not top ranked initially and is especially helpful for bugs that span multiple modules. From Figure 4 (b), we can observe that the divergence tracing generates more localization candidates and matched more vulnerable lines in this round.

**Detection Refinement.** The detection refinement begins based on crash logs and initial localization candidates. Each LLM-generated candidate gets a confidence label from very unlikely to very likely. We use these labels to sort and rank the candidates during the coarse-grained filtering. T2L agent selects code slices based on crash logs, stack trace information, and vulnerability patterns instead of exhaustively examining the full code snippet. In fine-grained refinement, we extract these slices and feed them back to the LLM along with the unified evidence, providing richer context surrounding the initially flagged locations.

The refinement process operates iteratively to help the agent correct early mistakes and discover vulnerabilities that are not immediately obvious from traces, particularly for complex vulnerabilities involving memory corruption where the crash point may be far away from the actual vulnerability. As Figure 4 (a) shows, based on the source code LLM interested, the refinement process successfully locates new lines that were not found during first step with only the runtime evidence.

### 4.3 T2L-ARVO BENCHMARK

The original ARVO dataset contains over 4,993 reproducible vulnerabilities across 250+ C/C++ projects, but its human-oriented design and build-centric structure do not directly support evaluating *agentic* vulnerability-localization systems. To bridge this gap, we developed T2L-ARVO, a 50-case benchmark derived from ARVO for assessing LLM agents in evidence-guided vulnerability localization, rather than human bug reproduction. T2L-ARVO maintains a moderate, calibrated difficulty while keeping the original crash-type diversity. Its construction employs a dual validation pipeline combining manual expert review with LLM-assisted checks to ensure each case is reproducible and appropriately challenging for automated agents. This yields a realistic, balanced benchmark for rigorous end-to-end evaluation of LLM-based vulnerability localization. In T2L-ARVO, each of the cases has multiple ground-truth bug lines. The average accuracy overall is denoted as  $Accuracy = \text{correct localized vulnerable lines} / \text{total vulnerable lines}$ .

**T2L-ARVO Composition** We analyzed 4,993 ARVO instances and grouped them by underlying failure mechanism: *Buffer Overflows* 49.9% ( $n = 2,490$ ), *Uninitialized Access & Unknown States* 35.4% ( $n = 1,768$ ), *Memory Lifecycle Errors* 11.5% ( $n = 573$ ), *Type Safety & Parameter Validation* 2.9% ( $n = 147$ ), and *System & Runtime Errors* 0.3% ( $n = 15$ ). Each family subsumes concrete subtypes (e.g., `heap-buffer-overflow`, `use-of-uninitialized-value`, `heap-use-after-free`, `bad-cast`). T2L-ARVO deliberately mirrors this distribution to avoid bias toward any single failure mode.

The final benchmark comprises 50 vulnerabilities samples, evenly sampled across five crash families (10 each) for broad yet controlled difficulty. Each family includes representative subtypes (e.g., `heap-buffer-overflow`, `use-of-uninitialized-value`, `heap-use-after-free`, `bad-cast`), covering single-file defects and cross-module interactions to prevent overflow bias and exercise diverse failure modes observed in real repositories.

**Verification Process** Our verification follows two main considerations. First, ARVO crash families contain multiple subtypes. To reduce redundancy and improve representativeness, we select only a few typical subtypes from each family (see A.1), preserving diversity without overloading the benchmark with similar vulnerabilities. Second, experts evaluate each sample’s complexity to keep T2L-ARVO at a moderate, calibrated difficulty. We score candidates using diff-based structural metrics (e.g., files changed, architectural spread, directory depth) and semantic factors (e.g., cross-module coupling, interface changes), ensuring that T2L-ARVO remains balanced in both coverage and difficulty.

Figure 5: Crash types in T2L-ARVO Bench.

Crash Family	Brief Description
Buffer Overflow	Violations of memory bounds (heap/stack)
Uninitialized Access	Reads from undefined or indeterminate state
Memory Lifecycle	Use-after-free / double-free / lifetime bugs
Type Safety	Bad casts, invalid args, contract violations
System Runtime	Environment and runtime interaction faults

## 5 EXPERIMENT SETUP

**Metrics.** We report two complementary scores for project-level OSS vulnerability studies. *Detection* asks whether the agent flags a vulnerability within the correct module/chunk and materially shrinks the search space. *Localization* requires exact line matches to ground-truth patches. Together, they separate “finding the neighborhood” from “pinpointing the line”, mirroring real debugging.

**Data Preparation.** We evaluate on the full T2L-ARVO set: 50 verified, structured challenges derived from ARVO, spanning diverse domains (e.g., imaging, networking) and balanced complexity so results reflect production patterns rather than a single project or bug family. Because ARVO lacks detection-ready chunking, T2L-ARVO adds AST-based segmentation: projects are partitioned into semantically meaningful units for scoring coarse *detection*, while exact line matches assess fine *localization*—a single framework for both levels.

**Model Selection.** We assess a set of state-of-the-art language models—both open-source and commercial to probe generality and robustness of T2L-Agent across architectures and scales, including open models such as Qwen3 Next, Qwen3 235B, DeepSeek 3.1, LLaMA 4 and commercial models like Claude4 Sonnet, GPT-5, GPT-4.1, GPT-4o-mini, Gemini 2.5 Pro, Gemini 2.5 Flash with a maximum budget \$1.0. We use API keys from commercial model’s official providers and Together.ai’s inference service for open source models.

Table 1: Base T2L-Agent Localization and Detection Rate Performance Across Different Models.

	% Avg.		Buffer		Initialize		Memory		Parameter		Runtime	
	Det	Loc										
GPT-5	44.3	<b>41.7</b>	57.5	<b>53.8</b>	35.6	35.5	<b>60.8</b>	<b>55.9</b>	36.5	39.4	11.2	10.0
GPT-4.1	<b>48.0</b>	38.5	<b>60.8</b>	36.5	50.6	<b>46.4</b>	<b>60.8</b>	46.1	26.5	29.9	21.2	<b>20.5</b>
GPT-4o-mini	44.3	22.6	<b>60.8</b>	20.2	48.1	12.5	55.8	24.7	28.7	26.7	11.2	10.0
Claude 4 Sonnet	45.9	30.5	57.5	50.6	<b>60.8</b>	36.1	1.3	31.6	<b>37.8</b>	<b>46.1</b>	<b>24.8</b>	0.5
Gemini2.5 Pro	17.4	10.5	25.0	5.6	25.0	20.0	11.3	10.8	5.4	11.9	10.0	10.5
Qwen3 235B	25.9	9.2	25.8	6.5	23.1	1.7	40.8	16.7	28.7	17.3	7.9	0.0
Qwen3 Next 80B	37.4	5.9	54.2	3.7	33.1	0.4	55.8	14.5	29.1	6.8	1.2	0.0

**Implementation.** We build T2L-Agent from scratch without LangChain, DSPy, and LlamaIndex to keep the core lightweight, retain fine-grained control over reasoning and tools, and maximize extensibility. The *Agentic Trace Analyzer* compiles targets with ASAN and collects crashes, stack traces, and allocation metadata to yield actionable traces for narrowing. Following ARVO’s layout, we maintain registry-backed, per-project environments and provide both vulnerable and patched revisions in containers. Our harness runs dockerized T2L-Agent’s that interface with T2L-ARVO via the Docker SDK for Python, orchestrating build-run-reproduce cycles from within the agent loop, ensuring consistent conditions, deterministic reproduction, and auditable measurement.

## 6 EVALUATION

### 6.1 BASELINE BENCHMARKING

We evaluate T2L-Agent on T2L-ARVO end-to-end and report Detection Rate and Localization Rate. Table 1 covers five models—GPT-5, GPT-4.1, GPT-4o-mini, Qwen 3 Next 80B, and Qwen3 235B, running under identical per-case budgets, environments, and AST-based chunking. Overall, detection is higher than localization by design. Under this setting, GPT-5 leads localization at 41.7%, followed by GPT-4.1 at 38.5%; GPT-4o-mini lands at 22.6%, and open-source models trail Qwen 3 235B 9.2% and Qwen 3 Next 80B 5.9%. For detection, GPT-4.1 is highest at 48.0%, with GPT-5 and GPT-4o-mini both at 44.3%; Qwen3 reaches 37.4% and Qwen3 235B 25.9%. While Gemini2.5 Pro shows limited effectiveness with 17.4% detection and 10.5% localization rates.

Family-wise patterns are consistent across metrics. *Buffer* and *Memory* are easier due to concrete runtime cues: for localization, GPT-5 reaches 53.8% and 55.9%, and most models cluster near the mid-50s for detection. *Initialize* sits mid-range and benefits from multi-step reasoning (e.g., GPT-5 35.5% vs. GPT-4.1 46.4% in localization). *Parameter* is often solvable from interface/call-site context (39.4% for GPT-5; 29.9% for GPT-4.1). *Runtime* remains uniformly hardest: detection hovers around 11.1–21.2% even for top configurations, and the best localization we observe is 20.5% on GPT-4.1, reflecting sparse, unstable traces.

Taken together, equal budgets surface clear profiles. GPT-5 (low-think) converts evidence into the strongest line-level localization, while GPT-4.1 extracts slightly more coarse-grained signal at the chunk level. GPT-4o-mini’s mix—competitive detection (44.3%) but weak localization (22.6%), suggests higher recall with looser ranking that does not always translate to precise line hits. Open-source models lag on both metrics under the same constraints, indicating gaps in code understanding and tool use rather than simple parameter tuning. Overall, improvements track the availability of concrete runtime evidence, and structured, tool-grounded reasoning appears more impactful than generation settings for end-to-end vulnerability localization.

### 6.2 DISCUSSION 1: FEATURE-WISE EVALUATION

**Agentic Trace Analyzer.** This table 2 demonstrates the critical effectiveness of our proposed Agentic Trace Analyzer (ATA) through ablation experiments. Without ATA, GPT-5 and Claude 4 Sonnet show 0.0% detection and localization across all families. This performance breakdown validates that our ATA component successfully bridges the gap between crash symptoms and vulnerability locations, addresses the fundamental challenge of vulnerability localization in complex codebases.

**Detection Refinement.** Compared with Tab 1, Tab. 2 shows broad, across-the-board gains after enabling refinement. Strong proprietary models improve steadily, while open-source models jump the most—Qwen3 235B’s localization rises by roughly sevenfold. Improvements vary by crash family: *Initialize* bugs benefit most (they demand multi-step reasoning), whereas *Buffer* and *Memory* see smaller lifts because concrete runtime evidence already anchors the search. *Runtime* cases remain hard—when traces are sparse, refinement offers limited benefit. Net effect: higher recall and more

Table 2: Localization and Detection Rate Performance of T2L-Agent with Refinement and Divergence Tracing Across Different Models.

	% Avg.				Buffer		Initialize		Memory		Parameter		Runtime	
	Det	Loc	$\Delta$ Det	$\Delta$ Loc	Det	Loc	Det	Loc	Det	Loc	Det	Loc	Det	Loc
w/ Detection Refinement														
GPT-5	52.4	44.5	+8.1 $\uparrow$	+2.8 $\uparrow$	57.5	55.0	55.6	43.1	60.8	41.3	43.5	48.2	21.2	20.5
GPT-4.1	48.3	40.8	+0.3 $\uparrow$	+2.3 $\uparrow$	60.8	51.9	53.1	44.9	57.5	46.1	39.8	42.9	6.7	0.0
GPT-4o-mini	34.6	29.1	-9.7 $\downarrow$	+6.5 $\uparrow$	45.8	43.6	30.6	20.2	45.8	33.6	22.4	21.2	0.0	0.0
Claude 4 Sonnet	44.8	41.4	-1.1 $\downarrow$	+10.9 $\uparrow$	57.5	54.3	40.6	43.2	61.7	52.5	26.1	29.4	14.6	10.5
Gemini2.5 Pro	14.1	11.4	-3.3 $\downarrow$	-0.9 $\downarrow$	10.0	8.6	20.0	16.2	40.0	32.1	0.4	0.3	0.0	0.0
Qwen3 Next 80B	42.9	39.5	+5.5 $\uparrow$	+33.6 $\uparrow$	60.8	55.1	50.6	44.4	60.8	48.6	25.7	29.2	0.0	0.0
Qwen3 235B	34.1	26.7	+8.2 $\uparrow$	+17.5 $\uparrow$	34.2	32.0	30.6	34.5	57.5	38.8	23.3	13.3	5.0	8.2
Gemini 2.5 Flash	22.5	18.4	-	-	34.2	0.6	40.8	25.7	7.9	27.6	0.4	33.4	24.6	0.6
Llama4	28.3	28.1	-	-	30.8	25.6	35.8	26.1	0.0	27.9	24.1	32.0	30.2	0.0
Deepseek V3.1	53.9	53.4	-	-	60.8	55.6	<b>62.5</b>	47.5	16.3	58.1	55.0	<b>60.2</b>	47.6	19.4
w/ Divergence Tracing														
GPT-5	<b>58.0</b>	52.0	+13.7 $\uparrow$	+10.3 $\uparrow$	60.8	56.8	60.6	53.4	<b>62.5</b>	47.6	53.2	46.7	26.2	<b>28.7</b>
GPT-4.1	52.0	49.9	+4.0 $\uparrow$	+11.4 $\uparrow$	60.8	53.5	60.6	<b>57.3</b>	57.5	53.0	43.2	37.1	21.2	20.1
GPT-4o-mini	47.2	43.3	+2.9 $\uparrow$	+20.7 $\uparrow$	<b>64.2</b>	55.6	55.6	48.8	52.5	47.0	33.9	32.4	1.2	0.5
Claude 4 Sonnet	48.7	49.8	+2.8 $\uparrow$	+19.3 $\uparrow$	60.8	55.6	<b>62.5</b>	39.8	11.3	57.5	<b>53.7</b>	57.6	46.3	10.6
Qwen 3 Next 80B	51.2	<b>54.8</b>	+13.8 $\uparrow$	+48.9 $\uparrow$	<b>64.2</b>	<b>58.1</b>	<b>62.5</b>	43.2	11.3	<b>63.2</b>	<b>58.6</b>	57.7	<b>48.8</b>	21.2
Qwen 3 235B	42.7	42.1	+16.8 $\uparrow$	+32.9 $\uparrow$	50.8	50.6	47.5	40.2	1.0	45.4	46.9	53.5	33.7	11.2
w/o Agentic Trace Analyzer														
GPT-5	0.0	0.0	-44.3 $\downarrow$	-41.7 $\downarrow$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Claude 4 Sonnet	0.0	0.0	-45.9 $\downarrow$	-30.5 $\downarrow$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

precise line-level hits with minimal tuning. Several additional models show promising performance. Deepseek V3.1 achieves the highest overall results with 53.9% detection and 53.4% localization rate. LLaMa 4 demonstrates balanced capabilities on both metrics, and Gemini 2.5 Flash shows variable performance across crash families.

**Divergence Tracing.** Tab. 2 shows divergence tracing yields the strongest gains. All models improve on both metrics: GPT-5 is up 13.7% in detection and 10.3 in localization; GPT-4.1 gains 4.0% and 11.4% respectively. Qwen3 Next see the largest jumps with adding 13.8% in detection and 48.9% in localization, while Qwen3 235B adds 16.8% and 32.9%. These consistent lifts across architectures highlight divergence tracing as a core algorithmic upgrade for vulnerability localization.

### 6.3 DISCUSSION 2: PARAMETER TUNING

**Thinking Budget.** As shown in Tab. 3, more thinking didn’t help. On GPT-5, the Medium budget outperforms High with 50.9% detection vs 41.3%, and 41.6% localization vs 36.1%. The Low setting trails Medium by a few points yet often matches or even exceeds High on key metrics, while sharply reducing compute and latency. This pattern suggests diminishing returns—and decision drag—at very high budgets: the model over-explores, delays commitment, and accumulates tool-use errors. In practice, Medium strikes the best accuracy–cost balance for vulnerability localization; Low is a strong option when throughput and responsiveness matter most.

Table 3: Localization and Detection Rate Performance of T2L-Agent for Temperature Tuning and Chain of Thought Across Different Models.

	Config	% Avg.		Buffer		Initialize		Memory		Parameter		Runtime	
		Det	Loc										
Temperature													
GPT-4.1	0.2	<b>51.0</b>	43.6	57.5	44.4	55.6	<b>51.1</b>	55.8	45.7	43.2	35.4	21.2	<b>20.2</b>
GPT-4.1	0.6	50.8	43.5	<b>60.8</b>	52.0	53.1	47.8	55.8	44.8	39.4	<b>42.7</b>	17.9	10.1
Claude 4 Sonnet	0.2	46.5	43.0	54.2	<b>56.1</b>	55.6	43.0	<b>61.7</b>	49.0	36.5	39.7	11.2	10.9
Claude 4 Sonnet	0.6	47.3	<b>44.9</b>	54.2	<b>56.1</b>	50.6	48.0	60.8	<b>53.6</b>	36.5	39.6	11.2	10.9
Reasoning Effect													
GPT-5	High	41.3	36.1	54.2	47.8	40.6	32.4	55.8	39.3	32.8	36.7	10.0	10.0
GPT-5	Medium	50.9	41.6	<b>60.8</b>	55.6	<b>60.8</b>	42.8	11.3	46.1	<b>45.8</b>	40.4	<b>47.4</b>	10.5

**Temperature.** Temperature changes barely matter from Tab. 3. On GPT-4.1, detection is 51.0% at 0.2 and 50.8% at 0.6, with localization 43.6% vs. 43.5%. Claude 4 Sonnet shows the same pattern: 46.5% vs. 47.3% detection and 43.0% vs. 44.9% localization. Performance is stable in 0.2-0.6 range. The exception is *Initialize* bugs, which are more temperature sensitive than *Buffer* and *Memory* cases that lean on concrete runtime evidence. Overall, precise localization benefits more from structured, tool-grounded reasoning than from extra sampling, making parameter choices simple.



---

## ETHICS STATEMENT

This work builds upon the ARVO dataset for vulnerability analysis. Our T2L-ARVO benchmark is constructed using its full-version data, and we have properly cited the original ARVO project to acknowledge its contribution and comply with copyright and attribution standards. No additional data collection, user studies, or ethically sensitive procedures involved in the construction or evaluation of T2L-Agent. All experiments were conducted on Linux servers with only open source dependencies, and our system does not involve any privacy-sensitive data, bias-sensitive decision-making, or potentially harmful applications. We also note that large language models were used solely for light editing and polishing of manuscript, with no involvement in system design, code generation, or experimental results. Their use was limited to improving readability and presentation clarity.

## REPRODUCIBILITY STATEMENT

To ensure reproducibility, we will publicly release the full T2L-Agent framework upon publish of this paper, including all module codebase, evaluation scripts, and benchmark data used in this paper. Our implementation does not rely on any proprietary components. Detailed descriptions of our methodology are provided in the main text (Sections 4.1, 4.3) and supported by step-by-step examples in the appendix. All experimental configurations, including model versions, prompting strategies, and budget constraints, will be documented and made available upon publication to enable full replication of our results.

## REFERENCES

- Talor Abramovich, Meet Udeshi, Minghao Shao, Kilian Lieret, Haoran Xi, Kimberly Milner, Sofija Jancheska, John Yang, Carlos E Jimenez, Farshad Khorrani, et al. Enigma: Interactive tools substantially assist llm agents in finding security vulnerabilities. In *Forty-second International Conference on Machine Learning*, 2025.
- Alsharif Abuadbbba, Chris Hicks, Kristen Moore, Vasilios Mavroudis, Burak Hasircioglu, Diksha Goel, and Piers Jennings. From promise to peril: Rethinking cybersecurity red and blue teaming in the age of llms. *arXiv preprint arXiv:2506.13434*, 2025.
- Hiralal Agrawal and Joseph R Horgan. Dynamic program slicing. *ACM SIGPlan Notices*, 25(6): 246–256, 1990.
- Moumita Asad, Rafed Muhammad Yasir, Armin Geramirad, and Sam Malek. Leveraging large language model for information retrieval-based bug localization. *arXiv preprint arXiv:2508.00253*, 2025.
- Abdul Basit, Nouhaila Innan, Muhammad Haider Asif, Minghao Shao, Muhammad Kashif, Alberto Marchisio, and Muhammad Shafique. Pennylang: Pioneering llm-based quantum code generation with a novel pennylane-centric dataset. *arXiv preprint arXiv:2503.02497*, 2025.
- Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.
- Roland Croft, M Ali Babar, and M Mehdi Kholoosi. Data quality for software vulnerability datasets. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 121–133. IEEE, 2023.
- CVE Details. Number of common it security vulnerabilities and exposures (cves) worldwide from 2009 to 2024 ytd. Statista, August 2024. URL <https://www.statista.com/statistics/500755/worldwide-common-vulnerabilities-and-exposures/>. Accessed: September 23, 2025.
- Dark Reading. Juniper research study reveals staggering cost of vulnerable software supply chains. Dark Reading, 2024. URL <https://www.darkreading.com/cybersecurity-operations/juniper-research-study-reveals-staggering-cost-of-vulnerable-software-supply-chains->. Accessed: September 23, 2025.

- 
- Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. {PentestGPT}: Evaluating and harnessing large language models for automated penetration testing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 847–864, 2024.
- Dinil Mon Divakaran and Sai Teja Peddinti. Llms for cyber security: New opportunities. *arXiv preprint arXiv:2404.11338*, 2024.
- Michael Fu and Chakkrit Tantithamthavorn. Linevul: A transformer-based line-level vulnerability prediction. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pp. 608–620, 2022. doi: 10.1145/3524842.3528452.
- GitHub, Inc. GitHub Copilot: Your AI pair programmer, 2021. URL <https://github.com/features/copilot>. Accessed: September 24 2025.
- Yuejun Guo, Seifeddine Bettaieb, and Fran Casino. Vulnerability analysis and detection using machine learning approaches, 2024. URL <https://link.springer.com/article/10.1007/s10207-024-00888-y>. DOI: 10.1007/s10207-024-00888-y.
- Indusface. 181 cybersecurity statistics for 2025. Indusface Blog, 2024. URL <https://www.indusface.com/blog/key-cybersecurity-statistics/>. Accessed: September 23, 2025.
- David Isern and Antonio Moreno. A systematic literature review of agents applied in healthcare. *Journal of medical systems*, 40(2):43, 2016.
- Zhonghao Jiang, Xiaoxue Ren, Meng Yan, Wei Jiang, Yong Li, and Zhongxin Liu. Cosil: Software issue localization via llm-driven code repository graph searching. *arXiv preprint arXiv:2503.22424*, 2025.
- Mingyu Jin, Haochen Xue, Zhenting Wang, Boming Kang, Ruosong Ye, Kaixiong Zhou, Mengnan Du, and Yongfeng Zhang. Prollm: protein chain-of-thoughts enhanced llm for protein-protein interaction prediction. *arXiv preprint arXiv:2405.06649*, 2024.
- Sungmin Kang, Gabin An, and Shin Yoo. A quantitative and qualitative evaluation of llm-based explainable fault localization. *Proceedings of the ACM on Software Engineering*, 1(FSE):1424–1446, 2024.
- Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2757–2791, 2025.
- Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024a.
- Yue Li, Xiao Li, Hao Wu, Yue Zhang, Xiuzhen Cheng, Sheng Zhong, and Fengyuan Xu. Attention is all you need for llm-based code vulnerability localization. *arXiv e-prints*, pp. arXiv–2410, 2024b.
- Jingchen Liu, Christos A. Makridis, and Evan Galinkin. Cybersecurity vulnerabilities and their financial impact. *VoxEU*, 2024. URL <https://cepr.org/voxeu/columns/cybersecurity-vulnerabilities-and-their-financial-impact>. Accessed: September 23, 2025.
- Xiaoqun Liu, Feiyang Yu, Xi Li, Guanhua Yan, Ping Yang, and Zhaohan Xi. Benchmarking llms in an embodied environment for blue team threat hunting. *arXiv preprint arXiv:2505.11901*, 2025.
- Ana Nunez, Nafis Tanveer Islam, Sumit Kumar Jha, and Peyman Najafirad. Autosafecoder: A multi-agent framework for securing llm code generation through static analysis and fuzz testing. *arXiv preprint arXiv:2409.10737*, 2024.
- Yihao Qin, Shangwen Wang, Yiling Lou, Jinhao Dong, Kaixin Wang, Xiaoling Li, and Xiaoguang Mao. Agentfl: Scaling llm-based fault localization to project-level context. *arXiv preprint arXiv:2403.16362*, 2024.

- 
- Md Nakhla Rafi, Dong Jae Kim, Tse-Hsun Chen, and Shaowei Wang. A multi-agent approach to fault localization via graph-based retrieval and reflexion. *arXiv preprint arXiv:2409.13642*, 2024.
- Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.
- Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E. Perry. Improving bug localization using structured information retrieval. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 345–355, 2013. doi: 10.1109/ASE.2013.6693093.
- Minghao Shao, Abdul Basit, Ramesh Karri, and Muhammad Shafique. Survey of different large language model architectures: Trends, benchmarks, and challenges. *IEEE Access*, 2024a.
- Minghao Shao, Boyuan Chen, Sofija Jancheska, Brendan Dolan-Gavitt, Siddharth Garg, Ramesh Karri, and Muhammad Shafique. An empirical evaluation of llms for solving offensive security challenges. *arXiv preprint arXiv:2402.11814*, 2024b.
- Minghao Shao, Sofija Jancheska, Meet Udeshi, Brendan Dolan-Gavitt, Kimberly Milner, Boyuan Chen, Max Yin, Siddharth Garg, Prashanth Krishnamurthy, Farshad Khorrami, et al. Nyu ctf bench: A scalable open-source benchmark dataset for evaluating llms in offensive security. *Advances in Neural Information Processing Systems*, 37:57472–57498, 2024c.
- Minghao Shao, Haoran Xi, Nanda Rani, Meet Udeshi, Venkata Sai Charan Putrevu, Kimberly Milner, Brendan Dolan-Gavitt, Sandeep Kumar Shukla, Prashanth Krishnamurthy, Farshad Khorrami, et al. Craken: Cybersecurity llm agent with knowledge-based execution. *arXiv preprint arXiv:2505.17107*, 2025.
- Xiangmin Shen, Lingzhi Wang, Zhenyuan Li, Yan Chen, Wencheng Zhao, Dawei Sun, Jiashui Wang, and Wei Ruan. Pentestagent: Incorporating llm agents to automated penetration testing. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, pp. 375–391, 2025.
- Francesco Sovrano, Adam Bauer, and Alberto Bacchelli. Large language models for in-file vulnerability localization can be “lost in the end”. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3715758. URL <https://doi.org/10.1145/3715758>.
- Adam Stein, Arthur Wayne, Aaditya Naik, Mayur Naik, and Eric Wong. Where’s the bug? attention probing for scalable fault localization. *arXiv preprint arXiv:2502.13966*, 2025.
- tmylla and contributors. Awesome-llm4cybersecurity: An overview of llms for cybersecurity. GitHub Repository, 2024. URL <https://github.com/tmylla/Awesome-LLM4Cybersecurity>. Accessed: September 23, 2025.
- Meet Udeshi, Minghao Shao, Haoran Xi, Nanda Rani, Kimberly Milner, Venkata Sai Charan Putrevu, Brendan Dolan-Gavitt, Sandeep Kumar Shukla, Prashanth Krishnamurthy, Farshad Khorrami, et al. D-cipher: Dynamic collaborative intelligent multi-agent system with planner and heterogeneous executors for offensive security. *arXiv preprint arXiv:2502.10931*, 2025.
- Shaowei Wang and David Lo. Version history, similar report, and structure: putting them together for improved bug localization. In *Proceedings of the 22nd International Conference on Program Comprehension, ICPC 2014*, pp. 53–63, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328791. doi: 10.1145/2597008.2597148. URL <https://doi.org/10.1145/2597008.2597148>.
- Zeng Wang, Lilas Alrahis, Likhitha Mankali, Johann Knechtel, and Ozgur Sinanoglu. Llms and the future of chip design: Unveiling security risks and building trust. In *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 385–390. IEEE, 2024.
- Mark Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering, ICSE ’81*, pp. 439–449. IEEE Press, 1981. ISBN 0897911466.

- 
- Cheng Weng, Yihao Qin, Bo Lin, Pei Liu, and Liqian Chen. Matsvd: Boosting statement-level vulnerability detection via dependency-based attention. In *Proceedings of the 15th Asia-Pacific Symposium on Internetware*, Internetware '24, pp. 115–124, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400707056. doi: 10.1145/3671016.3674807. URL <https://doi.org/10.1145/3671016.3674807>.
- Haoran Xue, Gias Uddin, and Song Wang. Pagent: Learning to patch software engineering agents. *arXiv preprint arXiv:2506.17772*, 2025.
- Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE symposium on security and privacy*, pp. 590–604. IEEE, 2014.
- Aidan ZH Yang, Claire Le Goues, Ruben Martins, and Vincent Hellendoorn. Large language models for test-free fault localization. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pp. 1–12, 2024a.
- Haoran Yang, Yu Nong, Tao Zhang, Xiapu Luo, and Haipeng Cai. Learning to detect and localize multilingual bugs. *Proc. ACM Softw. Eng.*, 1(FSE), July 2024b. doi: 10.1145/3660804. URL <https://doi.org/10.1145/3660804>.
- John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854, 2023.
- John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024c.
- Inseok Yeo, Duksan Ryu, and Jongmoon Baik. Improving llm-based fault localization with external memory and project context. *arXiv preprint arXiv:2506.03585*, 2025.
- Zheng Yu, Ziyi Guo, Yuhang Wu, Jiahao Yu, Meng Xu, Dongliang Mu, Yan Chen, and Xinyu Xing. Patchagent: A practical program repair agent mimicking human expertise. In *Proceedings of the 34th USENIX Security Symposium (USENIX Security'25)*, Seattle, WA, USA, 2025.
- Andy K Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, et al. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. *arXiv preprint arXiv:2408.08926*, 2024a.
- Jian Zhang, Chong Wang, Anran Li, Weisong Sun, Cen Zhang, Wei Ma, and Yang Liu. An empirical study of automated vulnerability localization with large language models. *arXiv preprint arXiv:2404.00287*, 2024b.
- Jie Zhang, Haoyu Bu, Hui Wen, Yongji Liu, Haiqiang Fei, Rongrong Xi, Lun Li, Yun Yang, Hong-song Zhu, and Dan Meng. When llms meet cybersecurity: a systematic literature review. *Cybersecurity*, 8(1):1–41, 2025a. doi: 10.1186/s42400-025-00361-w. URL <https://cybersecurity.springeropen.com/articles/10.1186/s42400-025-00361-w>.
- Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025b.
- Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *2012 34th International conference on software engineering (ICSE)*, pp. 14–24. IEEE, 2012.
- Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32, 2019.

## A APPENDIX

### A.1 ARVO CRASH TYPE

To understand where our vulnerability localization agent should focus, we analyze the crash type distribution in ARVO dataset. ARVO’s crash type distribution is dominated by classic memory corruption: Buffer Overflow accounts for 49.9% of crashes, led by heap-buffer-overflow (36.1% overall) and followed by stack-buffer-overflow (6.2%), index-out-of-bounds (3.3%), and global-buffer-overflow (3.2%), with underflows/containers in the long tail (around 1% each). Uninitialized Access & Unknown States is the second largest family at 35.4%, primarily use-of-uninitialized-value (20.3%), then UNKNOWN READ/WRITE (around 11.8% combined). Memory Lifecycle Errors contribute 11.5%, dominated by heap-use-after-free (7.8% overall) plus double-free, use-after-poison, and invalid frees. Type Safety & Parameter Validation is smaller (2.9%)—notably bad-cast (1.3%) and negative-size-param (0.8%). System & Runtime Errors are rare (0.3%). Overall, around 85% of ARVO crashes fall into Buffer Overflow or Uninitialized/Unknown categories.

Table 4: Crash Families and Subtypes Analysis

Family	Subtype	Count	% within family	% of total
<b>Buffer Overflow Vulnerabilities</b>				
	<i>Total</i>	2490	—	49.9%
	Heap-buffer-overflow	1802	72.4%	36.1%
	Stack-buffer-overflow	308	12.4%	6.2%
	Index-out-of-bounds	165	6.6%	3.3%
	Global-buffer-overflow	160	6.4%	3.2%
	Container-overflow	33	1.3%	0.7%
	Stack-buffer-underflow	13	0.5%	0.3%
	Dynamic-stack-buffer-overflow	9	0.4%	0.2%
<b>Uninitialized Access &amp; Unknown States</b>				
	<i>Total</i>	1768	—	35.4%
	Use-of-uninitialized-value	1015	57.4%	20.3%
	UNKNOWN READ	462	26.1%	9.3%
	Segv on unknown address	134	7.6%	2.7%
	UNKNOWN WRITE	123	7.0%	2.5%
	Null-dereference READ	25	1.4%	0.5%
	UNKNOWN	8	0.5%	0.2%
	Unknown-crash	1	0.1%	0.0%
<b>Memory Lifecycle Errors</b>				
	<i>Total</i>	573	—	11.5%
	Heap-use-after-free	389	67.9%	7.8%
	Heap-double-free	63	11.0%	1.3%
	Use-after-poison	48	8.4%	1.0%
	Invalid-free	29	5.1%	0.6%
	Stack-use-after-return	26	4.5%	0.5%
	Stack-use-after-scope	13	2.3%	0.3%
	Bad-free	5	0.9%	0.1%
<b>Type Safety &amp; Parameter Validation</b>				
	<i>Total</i>	147	—	2.9%
	Bad-cast	65	44.2%	1.3%
	Negative-size-param	42	28.6%	0.8%
	Memcpy-param-overlap	20	13.6%	0.4%
	Object-size	9	6.1%	0.2%
	Incorrect-function-pointer-type	6	4.1%	0.1%
	Non-positive-vla-bound-value	3	2.0%	0.1%
	Strcpy-param-overlap	1	0.7%	0.0%

Table 4: Crash Families and Subtypes Analysis (continued)

Family	Subtype	Count	% within family	% of total
	Strncpy-param-overlap	1	0.7%	0.0%
<b>System &amp; Runtime Errors</b>				
	<i>Total</i>	15	—	0.3%
	Check failed	6	40.0%	0.1%
	Unknown signal	6	40.0%	0.1%
	Bad parameters to <code>-sanitizer-annotate-contiguous-container</code>	2	13.3%	0.0%
	Nested bug in the same thread, aborting.	1	6.7%	0.0%

## A.2 ARVO DATASET PROFILING

We profiled the full ARVO corpus with all 4,993 vulnerabilities across 288 projects to guide T2L-ARVO’s design and document its coverage. The analysis maps distributional patterns, project traits, and crash-type frequencies, and clarifies how our 50-case subset aligns with the broader ARVO ecosystem. These profiles confirm that T2L-ARVO is representative across crash families, project complexity, and severity, providing a transparent baseline for extensions and alternative benchmarks. We include compact visualizations of these profiles to convey key ARVO factors—such as crash families, project complexity, and severity at a glance.

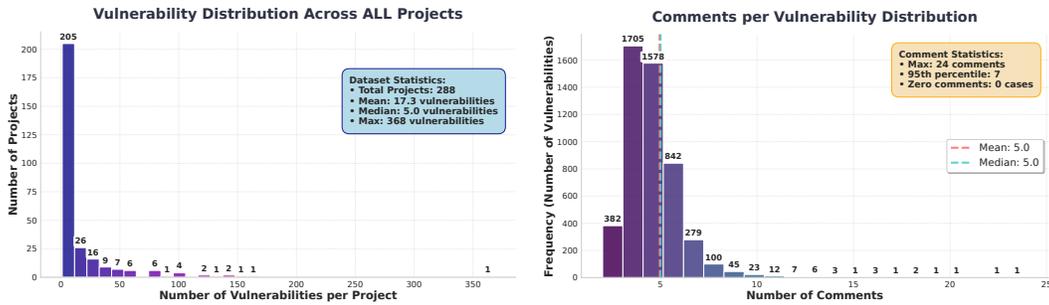


Figure 7: Distribution of vulnerability counts across the 288 ARVO projects. Most projects have under 25 vulnerabilities, with a long-tail of highly vulnerable ones.

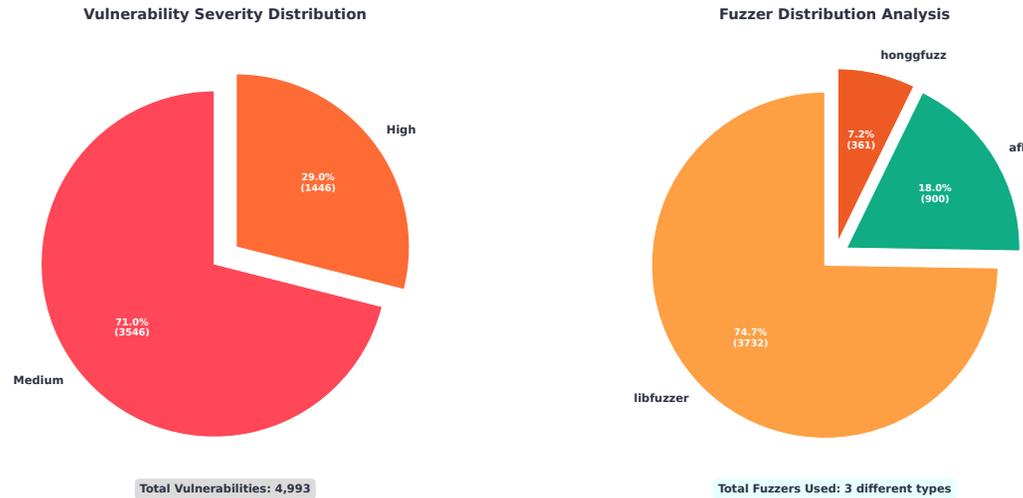


Figure 8: Breakdown of vulnerability severities in ARVO. Over 70% are medium severity, while high severity cases account for the remaining 29%.

### Top 30 Most Vulnerable Projects

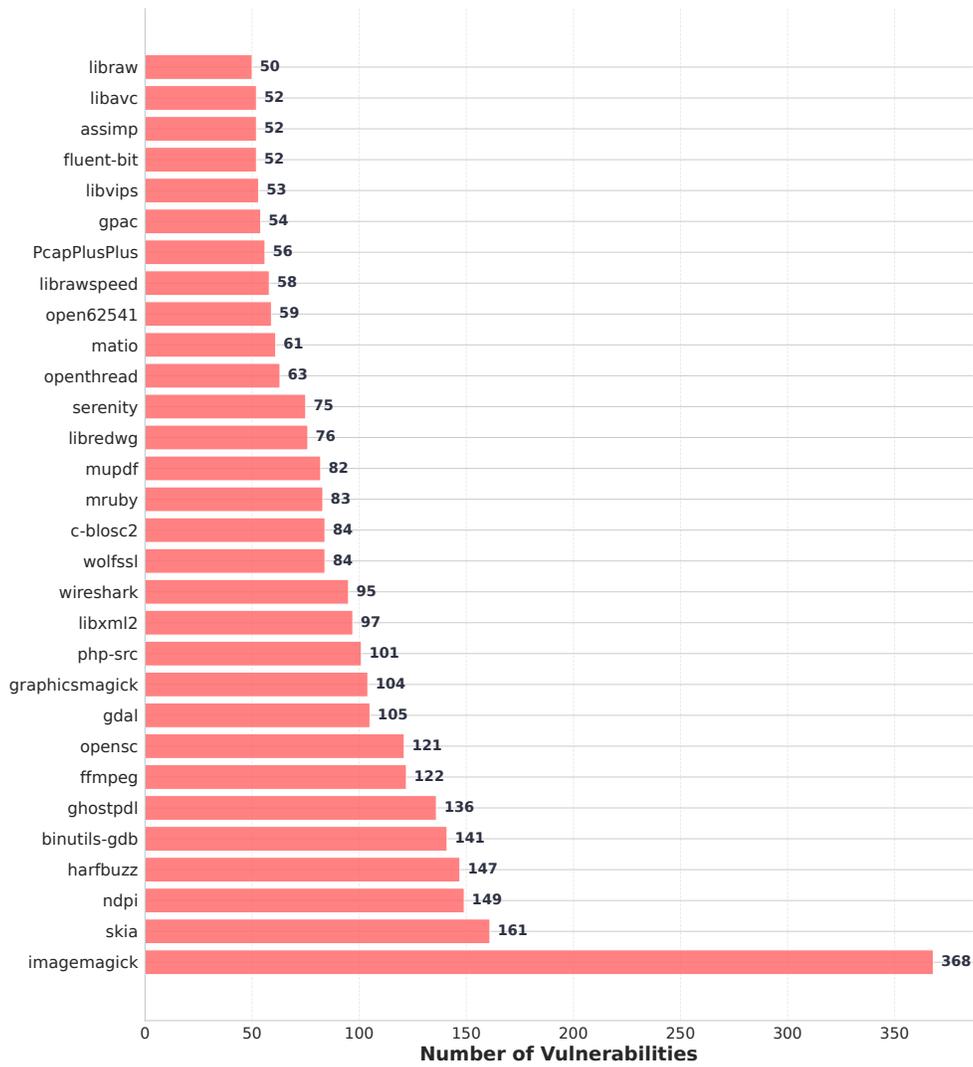


Figure 9: Analysis of fuzzing tools used in ARVO. libFuzzer dominates at 74.7%, followed by AFL and honggfuzz.

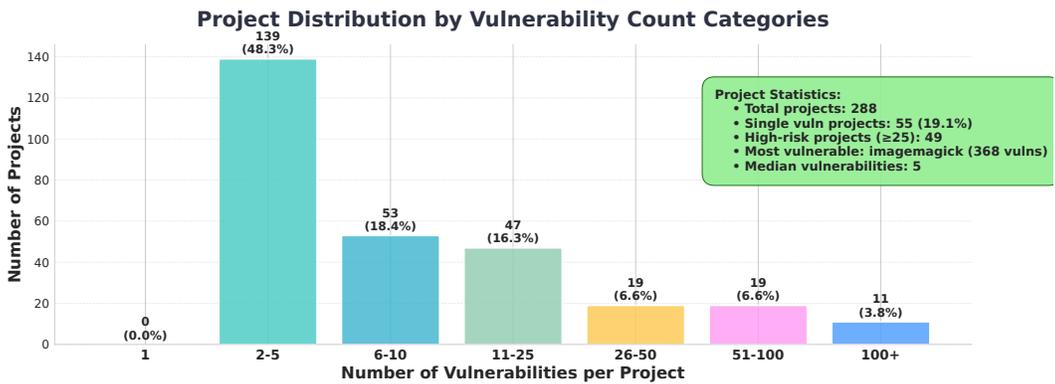


Figure 10: Visualization of the 30 most vulnerable projects in ARVO. ImageMagick leads with 368 vulnerabilities, with others showing diverse security footprints.

### Top 20 Most Common Crash Types

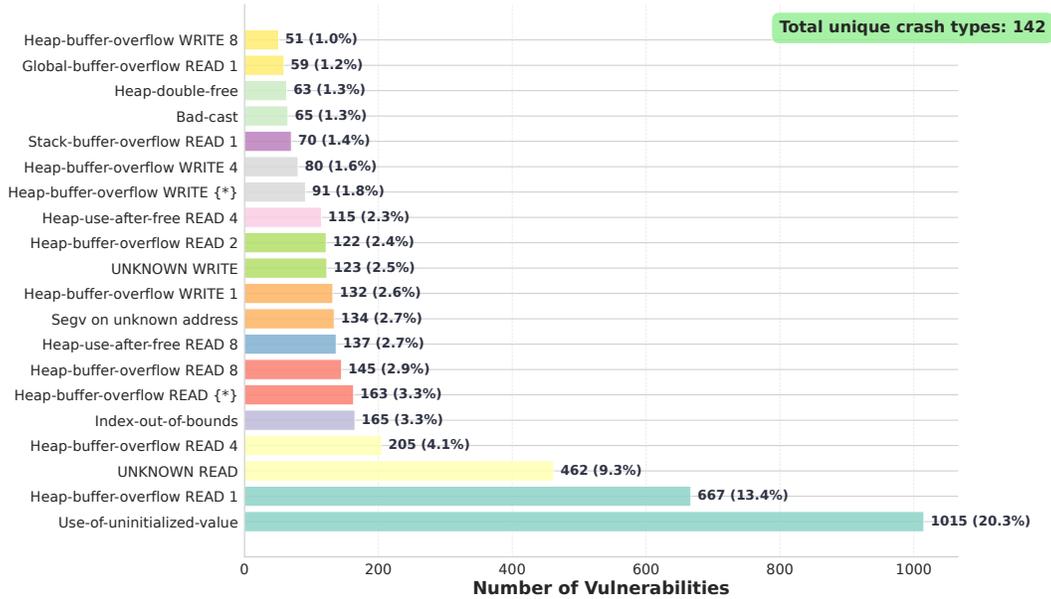


Figure 11: Project grouping by vulnerability count. Nearly half the projects have only 2–5 vulnerabilities, with very few exceeding 100.

### A.3 MODEL FAILURE ANALYSIS

We conducted a targeted failure analysis on several baseline models to map out common failure modes as shown in Tab. 12. Claude 4 Sonnet and Gemini 2.5 Pro hit the budget ceiling in 81.6% and 85.7% of runs respectively, indicating efficient resource utilization. GPT-5 reaches 61.2% with execution errors (28.6%), while Qwen3 235B struggles with basic data operations (59.2%). Open-source baselines stall early: Qwen 3 Next fails to surface actionable candidates in 44.9% of trials. Execution errors remain common across older models (20–30%), showing that tool use often breaks even when a plan exists. Net-net, while newer models show improved resource management, legacy models skew toward either incomplete exploration or difficulty navigating real-world code.

### Model Failure Analysis Among Common Failure Types

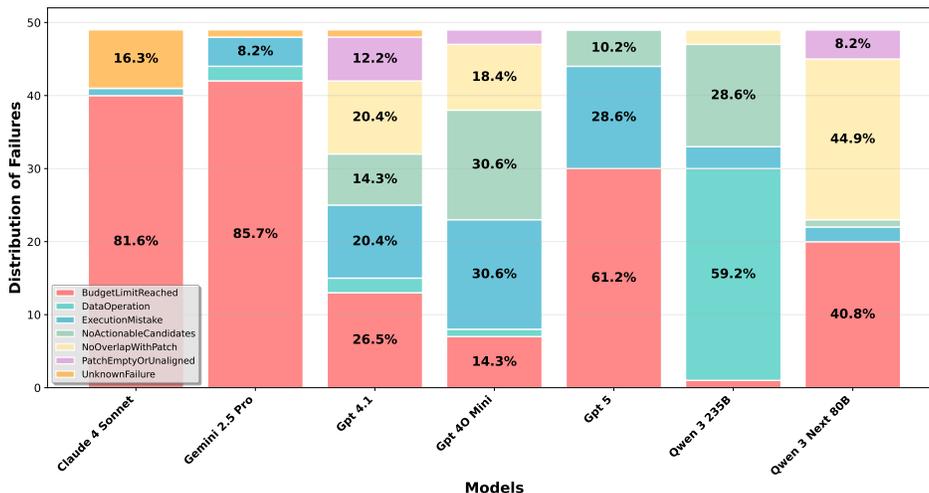


Figure 12: Model failure type distribution across five models on T2L-ARVO. GPT-5 commonly fails due to budget limits, while Qwen3 models often fail to generate actionable candidates.

#### A.4 T2L-ARVO CHALLENGE LIST

We provide the comprehensive list of T2L-ARVO benchmark we verified and collected in this work along with the key meta information for each challenge.

Table 5: Bug Analysis Results by Category

<b>Id</b>	<b>Fuzzer</b>	<b>Sanitizer</b>	<b>Project</b>	<b>Crash Type</b>	<b>Severity</b>
<b>System &amp; Runtime Errors</b>					
16737	libfuzzer	ubsan	graphicsmagick	Unknown signal	-
7966	libfuzzer	ubsan	graphicsmagick	Unknown signal	-
7654	libfuzzer	ubsan	graphicsmagick	Unknown signal	-
7639	libfuzzer	ubsan	graphicsmagick	Unknown signal	-
59193	libfuzzer	msan	faad2	Check failed	-
49915	libfuzzer	msan	ndpi	Check failed	-
48780	libfuzzer	msan	libvpx	Check failed	-
7361	afl	asan	ots	Bad parameters to sanitizer	-
32939	libfuzzer	asan	rdkit	Bad parameters to sanitizer	-
<b>Buffer Overflow Vulnerabilities</b>					
16614	libfuzzer	asan	openc	Heap-buffer-overflow	Med
13956	libfuzzer	asan	yara	Heap-buffer-overflow	Med
16615	libfuzzer	asan	openc	Heap-buffer-overflow	Med
20856	libfuzzer	asan	ndpi	Heap-buffer-overflow	Med
17330	libfuzzer	asan	openthread	Stack-buffer-overflow	High
42454	libfuzzer	asan	ghostpdl	Stack-buffer-overflow	High
17297	libfuzzer	asan	openthread	Stack-buffer-overflow	Med
18562	libfuzzer	asan	lwan	Global-buffer-overflow	-
30507	honggfuzz	asan	serenity	Global-buffer-overflow	-
18231	afl	asan	binutils-gdb	Global-buffer-overflow	-
<b>Uninitialized Access &amp; Unknown States</b>					
49493	libfuzzer	asan	mruby	Segv on unknown address	-
24290	honggfuzz	asan	libvips	Segv on unknown address	-
57037	libfuzzer	asan	mruby	Segv on unknown address	-
23778	libfuzzer	msan	binutils-gdb	Use-of-uninitialized-value	Med
20112	libfuzzer	msan	open62541	Use-of-uninitialized-value	Med
47855	libfuzzer	msan	harfbuzz	Use-of-uninitialized-value	Med
16857	libfuzzer	msan	matio	Use-of-uninitialized-value	Med
43989	libfuzzer	asan	ghostpdl	Null-dereference	-
2623	libfuzzer	asan	h2o	Null-dereference	-
45320	libfuzzer	asan	ghostpdl	Null-dereference	-
<b>Memory Lifecycle Errors</b>					
42503	libfuzzer	asan	php-src	Heap-use-after-free	High
38878	libfuzzer	asan	harfbuzz	Heap-use-after-free	High
14245	afl	asan	karchive	Heap-use-after-free	High
19723	libfuzzer	asan	leptonica	Heap-use-after-free	Med
33750	honggfuzz	asan	fluent-bit	Heap-double-free	High
34116	honggfuzz	asan	fluent-bit	Heap-double-free	High
20785	libfuzzer	asan	llvm-project	Use-after-poison	High
3505	afl	asan	librawspeed	Use-after-poison	High
51687	afl	asan	mongoose	Use-after-poison	High
31705	afl	asan	c-blosc2	Invalid-free	-
<b>Type Safety &amp; Parameter Validation</b>					
2798	libfuzzer	ubsan	gdal	Bad-cast	High

Continued on next page

Table 5 – continued from previous page

<b>Id</b>	<b>Fuzzer</b>	<b>Sanitizer</b>	<b>Project</b>	<b>Crash Type</b>	<b>Severity</b>
29267	libfuzzer	ubsan	serenity	Bad-cast	High
33150	libfuzzer	ubsan	libredwg	Object-size	Med
20217	libfuzzer	ubsan	arrow	Object-size	Med
12679	afl	asan	openthread	Memcpy-param-overlap	Med
23547	honggfuzz	asan	php-src	Memcpy-param-overlap	Med
25357	libfuzzer	asan	libsndfile	Negative-size-param	-
60605	libfuzzer	asan	ndpi	Negative-size-param	Med
2692	libfuzzer	ubsan	boringssl	Incorrect-function-pointer-type	Med
50623	libfuzzer	ubsan	serenity	Non-positive-vla-bound-value	Med

## A.5 T2L TOOLKIT LIST

We list the tools used in T2L-Agen and their roles in the analysis workflow. The framework is modular and can be easily extended with new tools based on task requirements.

Table 6: T2L Toolkit list and the usage description.

<b>Tool (NAME)</b>	<b>Description</b>
<code>view_source</code>	Preview a source file with line numbers. Optionally specify <code>start_line/end_line</code> .
<code>grep_source</code>	Search code by regex under a root directory. Returns <code>'file:line:match'</code> lines.
<code>insert_print</code>	Insert a single line of debug print before the given line number in a source file.
<code>build_project</code>	Build the project inside container. Default <code>workdir=/src</code> .
<code>container_exec</code>	Run an arbitrary shell command inside the container (advanced use).
<code>copy_out</code>	Copy a file/dir from container to host.
<code>giveup</code>	Give up this case to terminate it immediately. Use this to stop solving the ARVO container.
<code>diff_index</code>	Parse unified diff and build a simple line-level index without extra anchoring. Output JSON has per-file <code>{anchors_old, insert_points=[], per_line:{line-&gt;{roles, matched}}}</code> ; all line numbers are OLD-file coordinates from the diff.
<code>static_analysis</code>	Run comprehensive analysis. For binaries: run Ghidra RE then static tools; for sources: run static tools directly. Uses <code>cppcheck/clang-tidy/infer</code> ; aggregates findings to JSON and env state.
<code>chunk_case</code>	Parse C/C++ sources under <code>root_dir</code> with <code>tree-sitter</code> and save chunks JSON ( <code>index, file_path, chunk_kind, symbol, start/end_line, source, ast_type, imports</code> ).
<code>publish_verified_locations</code>	Verify/overwrite LLM locations by matching <code>symbol+snippet</code> in numbered snapshots; fallback to original lines; save verified JSON and snapshot index; updates <code>env..state['last_llm_json_path']</code> .
<code>run_san</code>	Run ARVO workflow that triggers ASAN/fuzzer and capture output.
<code>run_gdb</code>	Run gdb in the ARVO container and return a backtrace.
<code>llm_analyze</code>	Send crash/ASAN log to the LLM to predict likely bug locations (JSON expected). Supports refine mode with source slices.
<code>mark_diff</code>	Load anchored diff ( <code>anchors_old + insert_points</code> ) and mark chunks touched by these lines; updates chunks JSON (adds <code>diff, diff_hit_lines</code> ).
<code>compare_llm_metrics</code>	Compute detection rate (chunk-level) and localization rate (diff-line-level), plus strict localization by exact interval equality; updates JSON flags accordingly.
<code>gdb_script</code>	Run GDB non-interactively with <code>-batch</code> and provided commands; returns GDB output.
<code>extract_json</code>	Extract JSON array from a raw LLM response; optionally merge with last predictions; writes temp JSON and updates state.
<code>extract_modified_lines</code>	Parse a unified diff and return modified (file, line) list.
<code>compare_patch</code>	Compare LLM predicted spans with the patch; set solved by match rate.
<code>pipeline</code>	End-to-end: ASAN, GDB, LLM analyze, (optional) compare with patch.
<code>delegate</code>	Delegate a task to an executor LLM agent (autonomous, equipped for CTF-style tasks).
<code>static_analysis_config</code>	Enable/disable static analysis inclusion in crash log ( <code>enabled</code> flag).
<code>baseline_llm_analyze</code>	Single-shot baseline without ASAN/GDB or static context—ask LLM to guess once (no refine/verify/postprocess).











```
Prompt

Task: Orchestrate ARVO bug localization and evaluation.

Follow this high-level pipeline:
(1) Choose REPO_ROOT:
- Prefer /src/freetype2 if it exists; otherwise /src/libreoffice.
(2) chunk_case(root_dir=REPO_ROOT, out_json_path="/case_chunks.json")
(3) diff_index(diff_path, out_json_path="/diff_index.json")
(4) mark_diff(diff_json_path="/diff_index.json",
            chunks_json_path="/case_chunks.json")
(5) run_sanit, parse binary path from sanitized ASAN output
(6) run_gdb(binary_path=...) ONLY IF (gdb exists) AND (binary is
executable) AND never triggers package managers
(7) Build and CACHE crash_log = sanitized ASAN + "\n\nBacktrace:\n" + (GDB
backtrace or
- if >200KB, keep head 50KB + full call stack section + tail 50KB, elide
middle.
(8) DELEGATE to the Executor. The delegated task MUST ask for:
- llm_analyze(crash_log=...) - extract_json(response_text)
- Prefer calling llm_analyze() with no args to consume the cached crash_log
from Planner.
- Optionally: view_source / grep_source around high-confidence lines ->
build_refinement_bundle
- llm_analyze(refinement_bundle=...) - extract_json(ONE pass)
- If available: publish_verified_locations(..., repo_root=REPO_ROOT)
- compare_llm_metrics(diff_index_json_path="/diff_index.json",
llm_json_path=verified or raw JSONs,
chunks_json_path="/case_chunks.json",
line_stack=0)
- Finally: finish_task(summary=<<include metrics + paths>>)

Output only tool calls or delegated tasks as needed. Keep text minimal.
```

Thought

Planner note

```
Observation

You are the PLANNER for ARVO bug localization. You orchestrate the pipeline, call "lightweight
reasoning" tools and runtime collection (ASAN/GDB) yourself, and DELEGATE heavy
reasoning/evaluation to the Executor via delegate(task=...).

## Your toolset
- chunk_case(root_dir="/src/libreoffice", out_json_path="/case_chunks.json", include_context=0,
max_line_width=1000)
- diff_index(diff_path, out_json_path="/diff_index.json")
- mark_diff(diff_json_path="/diff_index.json", chunks_json_path="/case_chunks.json")
- run_sanit
- run_gdb(binary_path="/vuln")
- delegate(task="<string>") -> grep()

## Responsibilities
- OWN orchestration and state: start container once, choose REPO_ROOT, build chunks & diff index
once, mark diff on chunks once (unless inputs changed).
- YOU run ASAN/GDB and cache crash_log in environment state.
- For LL reasoning & evaluation: DELEGATE to the Executor with a clear, structured task that includes:
- What to analyze (use cached crash_log = llm_analyze -> extract_json).
- Which files to use (paths to diff_index.json, case_chunks.json)
- Whether to verify (publish_verified_locations if available) => compare_llm_metrics.
- Output contract (return metrics and paths in "finish_task(summary=...)", ).
- Aggregate results. -> Do not call (giveup!) after a single delegation. + Prefer at least 2-3 refinement
rounds unless clearly hopeless.

## Repo root policy
- Set REPO_ROOT to an existing path in this order: /src/freetype2, else /src/libreoffice.
- Use that REPO_ROOT consistently when asking the Executor to verify locations.

## Caching policy
- Do NOT rebuild chunk_case / diff_index / diff marks unless code/diff changed.
- Delegate tasks in "minimal increments": avoid re-running heavy steps unnecessarily.
```

Thought

Planner note

```
Observation

Task: Orchestrate ARVO bug localization and evaluation.

Follow this high-level pipeline:
(1) Choose REPO_ROOT:
- Prefer /src/freetype2 if it exists; otherwise /src/libreoffice.
(2) chunk_case(root_dir=REPO_ROOT, out_json_path="/case_chunks.json")
(3) diff_index(diff_path, out_json_path="/diff_index.json")
(4) mark_diff(diff_json_path="/diff_index.json", chunks_json_path="/case_chunks.json")
(5) run_sanit, parse binary path from sanitized ASAN output
(6) run_gdb(binary_path=...) ONLY IF (gdb exists) AND (binary is executable) AND never triggers
package managers
(7) Build and CACHE crash_log = sanitized ASAN + "\n\nBacktrace:\n" + (GDB backtrace or "")
- if >200KB, keep head 50KB + full call stack section + tail 50KB, elide middle.
(8) DELEGATE to the Executor. The delegated task MUST ask for:
- llm_analyze(crash_log=...) -> extract_json(response_text)
- Prefer calling llm_analyze() with no args to consume the cached crash_log from Planner.
- Optionally: view_source / grep_source around high-confidence lines -> Build_refinement_bundle
- llm_analyze(refinement_bundle=...) -> extract_json(ONE pass)
- If available: publish_verified_locations(..., repo_root=REPO_ROOT)
- compare_llm_metrics(diff_index_json_path="/diff_index.json",
llm_json_path=verified or raw JSONs,
chunks_json_path="/case_chunks.json",
line_stack=0)
- Finally: finish_task(summary=<<include metrics + paths>>)

Output only tool calls or delegated tasks as needed. Keep text minimal.
```

Thought

Planner call: chunk\_case

```
$ chunk_case(root_dir="/src/libreoffice",
out_json_path="/home/bug_localization_agents/work_arvo/20112
case_chunks.json")
```

Thought

Planner result: chunk\_case

```
Observation

{"ok": true, "root_dir": "/src/libreoffice", "json_path":
"/home/bug_localization_agents/work_arvo/20112/case_chunks.json", "num_files": 472, "num_chunks":
526}
```

Thought

Planner call: diff\_index

```
$ diff_index(diff_path="/home/bug_localization_agents/ARVO-Meta/patches
20112.diff",
out_json_path="/home/bug_localization_agents/work_arvo/20112
diff_index.json")
```

Thought

Planner result: diff\_index

```
Observation

{"ok": true, "json_path": "/home/bug_localization_agents/work_arvo/20112/diff_index.json", "num_files":
1, "num_chunks": 0, "num_per_line": 26}
```

Thought

Planner call: mark\_diff

```
$ mark_diff(diff_json_path="/home/bug_localization_agents/work_arvo/20112
diff_index.json",
chunks_json_path="/home/bug_localization_agents/work_arvo/20112
case_chunks.json")
```

Thought

Planner result: mark\_diff

```
Observation

{"ok": true, "chunks_json_path": "/home/bug_localization_agents/work_arvo/20112/case_chunks.json",
"num_chunks": 5265, "num_chunks_with_diff": 2}
```

Thought

Planner call: run\_san

```
$ run_san()
```

Thought

Planner result: run\_san

```
Observation

{"ok": true, "asan_output": {"ok": true, "asan_output": {"INFO: Seed: 2174243374\nINFO: Loaded 1
modules (10308 in-line, 2022a38, 0xa4ae78), \n\n/out/fuzz_json_decode: Running 1 inputs 1 time(s)
each.\n\nRunning: /tmp/poc/vuln/u001b[1m\u001b[31m\u001b[33m\u001b[35m\u001b[37m\u001b[39m\u001b[40m
0x6f4e49 in jmpOverRec
/src/open62541/src/ua_types_encoding_json.c:2339:16\n #1 0x6f40ed in jmpOverRec
/src/open62541/src/ua_types_encoding_json.c:2337:16\n #2 0x6e8bcb in jmpOverObject
/src/open62541/src/ua_types_encoding_json.c:2838:13\n #3 0x6f5da7 in Array_decodeJson_internal
/src/open62541/src/ua_types_encoding_json.c:3142:15\n #4 0x6d8ea7 in decodeFields
/src/open62541/src/ua_types_encoding_json.c:3096:23\n #5 0x6eb7bc in Variant_decodeJson
/src/open62541/src/ua_types_encoding_json.c:3321:11\n #6 0x6ed5d4 in LLWfuzzerTestOneInput
fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:554:15\n #7 0x445381 in
fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:292:49\n #8 0x44b09e in
fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerMain.cpp:775:9\n #9 0x47422 in main
/lib/libc.so.6:4-linux-gnu/libc.so.6:0x7f4e6c9283f in _libc_start_main
/out/fuzz_json_decode+0x41e828\n #15 0x41e828 in _start
/out/fuzz_json_decode+0x41e828\n #16 0x41e828 in _start
jmpOverRec-jmpOverRec-jmpOverRec\n u001b[1m\u001b[31m\u001b[33m\u001b[35m\u001b[37m\u001b[39m\u001b[40m
uninitialized value was created by a heap allocation\n u001b[1m\u001b[31m\u001b[33m\u001b[35m\u001b[37m\u001b[39m\u001b[40m
0x6d176 in UA_decodeJson /src/open62541/src/ua_types_encoding_json.c:296:39\n #3 0x524bb4 in
LLWfuzzerTestOneInput /src/open62541/tests/fuzz/fuzz_json_decode.cc:26:28\n #4 0x45ab51 in
fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:554:15\n #5 0x445381 in
fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:292:49\n #6 0x44b09e in
fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerMain.cpp:775:9\n #7 0x47422 in main
/lib/libc.so.6:4-linux-gnu/libc.so.6:0x7f4e6c9283f in _libc_start_main
/out/fuzz_json_decode+0x41e828\n #15 0x41e828 in _start
/out/fuzz_json_decode+0x41e828\n #16 0x41e828 in _start
malloc-UA memoryManager_malloc-UA decodeJson\n SUMMARY: MemorySanitizer:
use-of-uninitialized-value /src/open62541/src/ua_types_encoding_json.c:255:16 in
jmpOverRec\n unique heap origins: 2\n in use of, depth allocated bytes: 192\n unique origin
histories: 3\n in history depth: allocated bytes: 72\n in Exiting", "binary_path": "/out/fuzz_json_decode",
"cid": "98b0a0a6c72"}}
```

Thought

Planner call: run\_gdb

```
$ run_gdb(binary_path="/out/fuzz_json_decode")
```

Thought

Planner result: run\_gdb

```
Observation

{"ok": true, "warning": "Failed to install gdb inside the container."}
```

Thought

Planner call: delegate

```
$ delegate(task="llm_analyze(crash_log=INFO: Seed: 2174243374\nINFO: Loaded 1
modules (10308 in-line, 8-bit counters): 10308 [0xa221f0,
0xa22241], \n\nINFO: Loaded 1 PC tables (10308 PCs): 10308
[0xa22a38, 0xa4ae78], \n\n/out/fuzz_json_decode: Running 1 inputs 1 time(s)
each.\n\nRunning: /tmp/poc/vuln/u001b[1m\u001b[31m\u001b[33m\u001b[35m\u001b[37m\u001b[39m\u001b[40m
0x6f4e49 in jmpOverRec
/src/open62541/src/ua_types_encoding_json.c:2339:16\n #1 0x6f40ed in jmpOverRec
/src/open62541/src/ua_types_encoding_json.c:2337:16\n #2 0x6e8bcb in jmpOverObject
/src/open62541/src/ua_types_encoding_json.c:2838:13\n #3 0x6f5da7 in Array_decodeJson_internal
/src/open62541/src/ua_types_encoding_json.c:3142:15\n #4 0x6d8ea7 in decodeFields
/src/open62541/src/ua_types_encoding_json.c:3096:23\n #5 0x6eb7bc in Variant_decodeJson
/src/open62541/src/ua_types_encoding_json.c:\n #6 0x6ed5d4 in
LLWfuzzerTestOneInput /src/open62541/src/ua_types_encoding_json.c:3321:11\n #7
0x524bb4 in LLWfuzzerTestOneInput
/src/open62541/tests/fuzz/fuzz_json_decode.cc:26:28\n #8 0x445ab51 in
fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:554:15\n #9
0x445381 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned
long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:292:49\n #10
0x44b09e in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
const*, unsigned long))
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerMain.cpp:775:9\n #11
0x7f4e6c9283f in _libc_start_main
/lib/libc.so.6:4-linux-gnu/libc.so.6:0x7f4e6c9283f in _libc_start_main
/out/fuzz_json_decode+0x41e828\n #15 0x41e828 in _start
/out/fuzz_json_decode+0x41e828\n #16 0x41e828 in _start
jmpOverRec-jmpOverRec-jmpOverRec\n u001b[1m\u001b[31m\u001b[33m\u001b[35m\u001b[37m\u001b[39m\u001b[40m
uninitialized value was created by a heap allocation\n u001b[1m\u001b[31m\u001b[33m\u001b[35m\u001b[37m\u001b[39m\u001b[40m
0x7086cb in UA_memoryManager_malloc
/src/llvm/projects/compiler-rt/lib/msan/msan_interceptors.cpp:916:3\n #1
0x7086cb in UA_memoryManager_malloc
/src/open62541/tests/fuzz/custom_memory_manager.c:132:18\n #2 0x6ed176 in
UA_decodeJson /src/open62541/src/ua_types_encoding_json.c:296:39\n #3 0x524bb4 in
LLWfuzzerTestOneInput /src/open62541/tests/fuzz/fuzz_json_decode.cc:26:28\n #4 0x45ab51 in
fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:554:15\n #5 0x445381 in
fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:292:49\n #6 0x44b09e in
fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerMain.cpp:775:9\n #7 0x47422 in main
/lib/libc.so.6:4-linux-gnu/libc.so.6:0x7f4e6c9283f in _libc_start_main
/out/fuzz_json_decode+0x41e828\n #15 0x41e828 in _start
/out/fuzz_json_decode+0x41e828\n #16 0x41e828 in _start
malloc-UA memoryManager_malloc-UA decodeJson\n SUMMARY: MemorySanitizer:
use-of-uninitialized-value /src/open62541/src/ua_types_encoding_json.c:255:16 in
jmpOverRec\n unique heap origins: 2\n in use of, depth allocated bytes: 192\n unique origin
histories: 3\n in history depth: allocated bytes: 72\n in Exiting", "binary_path": "/out/fuzz_json_decode",
"cid": "98b0a0a6c72"}}
```



```
!confident: 'unlikely', 'reason': 'Main entry point handles overall fuzzer lifecycle; indirect impact on memory allocation practices.', 'symbol': 'main', 'id': 'json_path'}
/home/bug/localization_agents/work_arvo/20112/lm_batch_0001.json
```

Thought

```
Executor#0 call: compare_llm_metrics
$ compare_llm_metrics(llm_json_path="/home/bug/localization_agents/work_arvo_20112/lm_batch_0001.json",
chunks_json_path="/home/bug/localization_agents/work_arvo/20112/case_chunks.json",
diff_index_json_path="/home/bug/localization_agents/work_arvo/20112/diff_index.json",
line_slack=0)
```

Thought

Executor#0 result: compare\_llm\_metrics

Observation

```
{'ok': true, 'detection_rate': 1.0, 'localization_rate': 0.5769230769230769, 'strict_localization_rate': 0.0, 'inputs': {'llm_json_path': '/home/bug/localization_agents/work_arvo/20112/lm_batch_0001.json', 'chunks_json_path': '/home/bug/localization_agents/work_arvo/20112/case_chunks.json', 'diff_index_json_path': '/home/bug/localization_agents/work_arvo/20112/diff_index.json'}, 'detection': {'denom_diff_chunks': 2, 'num_defected_and_diff': 2, 'rate': 1.0, 'updated_chunks': true}, 'localization': {'total_lines': 26, 'matched_lines': 15, 'newly_matched': 15, 'rate': 0.5769230769230769, 'strict_localization': {'total_groups': 9, 'matched_groups': 0, 'rate': 0.0, 'saved_snapshot_error': 'Compare history'}}
```

Thought

Executor#0 call: view\_source

```
$ view_source(file_path="/src/open62541/src/ua_types_encoding_json.c",
start_line=235, end_line=238, max_bytes=20000)
```

Thought

Executor#0 result: view\_source

Observation

```
{'ok': true, 'file_path': '/src/open62541/src/ua_types_encoding_json.c', 'start_line': 235, 'end_line': 238, 'content': '\n235 else\n\n236 /* Only Primitive or string */\n237 parseCtx->index\n\n238 /*\n\n239 UA_Int16 oldIndex = parseCtx->index;\n\n240 /* Save index for later restore */\n241 int308\n\n242 /*\n\n243 UA_Int16 depth = 0;\n\n244 UA_StatusCode ret = searchObjectByRecursion(ctx,\n\n245 parseCtx, resultIndex, depth);\n\n246 /* Function used to jump over an object which cannot be parsed\n\n247 /*\n\n248 UA_Int16 depth = 0;\n\n249 UA_StatusCode ret = UA_STATUSCODE_GOOD;\n\n250 /*\n\n251 CHECK_TOKEN_BOUNDS(int322 int323 /* parseCtx->tokenArray[parseCtx->index].type ==\n\n252 ISMN_OBJECT */\n\n253 size_t objectCount =\n\n254 size_t *parseCtx->tokenArray[parseCtx->index].size;\n\n255 int326 parseCtx->index++; /* Object to\n\n256 first key? */\n\n257 CHECK_TOKEN_BOUNDS(int328 int329 /* int3230 for(i = 0; i < objectCount; i++)\n\n258 /*\n\n259 int331 CHECK_TOKEN_BOUNDS(int332 int333 parseCtx->index++; /* Value */\n\n260 int334\n\n261 ISMN_OBJECT */\n\n262 int335 jumpOverRec(parseCtx, resultIndex, (UA_Int16)depth + 1));\n\n263 /*\n\n264 if(parseCtx->tokenArray[parseCtx->index].type == ISMN_ARRAY)\n\n265 int336 jumpOverRec(parseCtx, resultIndex, (UA_Int16)depth + 1);\n\n266 /*\n\n267 parseCtx->index++; /*\n\n268 int337 /*\n\n269 int338 /*\n\n270 parseCtx->index++; /*\n\n271 int339 /*\n\n272 int340 /*\n\n273 int341 /*\n\n274 int342 /*\n\n275 int343 /*\n\n276 int344 /*\n\n277 int345 /*\n\n278 int346 /*\n\n279 int347 /*\n\n280 int348 /*\n\n281 int349 /*\n\n282 int350 /*\n\n283 int351 /*\n\n284 int352 /*\n\n285 int353 /*\n\n286 int354 /*\n\n287 int355 /*\n\n288 int356 /*\n\n289 int357 /*\n\n290 int358 /*\n\n291 int359 /*\n\n292 int360 /*\n\n293 int361 /*\n\n294 int362 /*\n\n295 int363 /*\n\n296 int364 /*\n\n297 int365 /*\n\n298 int366 /*\n\n299 int367 /*\n\n300 int368 /*\n\n301 int369 /*\n\n302 int370 /*\n\n303 int371 /*\n\n304 int372 /*\n\n305 int373 /*\n\n306 int374 /*\n\n307 int375 /*\n\n308 int376 /*\n\n309 int377 /*\n\n310 int378 /*\n\n311 int379 /*\n\n312 int380 /*\n\n313 int381 /*\n\n314 int382 /*\n\n315 int383 /*\n\n316 int384 /*\n\n317 int385 /*\n\n318 int386 /*\n\n319 int387 /*\n\n320 int388 /*\n\n321 int389 /*\n\n322 int390 /*\n\n323 int391 /*\n\n324 int392 /*\n\n325 int393 /*\n\n326 int394 /*\n\n327 int395 /*\n\n328 int396 /*\n\n329 int397 /*\n\n330 int398 /*\n\n331 int399 /*\n\n332 int400 /*\n\n333 int401 /*\n\n334 int402 /*\n\n335 int403 /*\n\n336 int404 /*\n\n337 int405 /*\n\n338 int406 /*\n\n339 int407 /*\n\n340 int408 /*\n\n341 int409 /*\n\n342 int410 /*\n\n343 int411 /*\n\n344 int412 /*\n\n345 int413 /*\n\n346 int414 /*\n\n347 int415 /*\n\n348 int416 /*\n\n349 int417 /*\n\n350 int418 /*\n\n351 int419 /*\n\n352 int420 /*\n\n353 int421 /*\n\n354 int422 /*\n\n355 int423 /*\n\n356 int424 /*\n\n357 int425 /*\n\n358 int426 /*\n\n359 int427 /*\n\n360 int428 /*\n\n361 int429 /*\n\n362 int430 /*\n\n363 int431 /*\n\n364 int432 /*\n\n365 int433 /*\n\n366 int434 /*\n\n367 int435 /*\n\n368 int436 /*\n\n369 int437 /*\n\n370 int438 /*\n\n371 int439 /*\n\n372 int440 /*\n\n373 int441 /*\n\n374 int442 /*\n\n375 int443 /*\n\n376 int444 /*\n\n377 int445 /*\n\n378 int446 /*\n\n379 int447 /*\n\n380 int448 /*\n\n381 int449 /*\n\n382 int450 /*\n\n383 int451 /*\n\n384 int452 /*\n\n385 int453 /*\n\n386 int454 /*\n\n387 int455 /*\n\n388 int456 /*\n\n389 int457 /*\n\n390 int458 /*\n\n391 int459 /*\n\n392 int460 /*\n\n393 int461 /*\n\n394 int462 /*\n\n395 int463 /*\n\n396 int464 /*\n\n397 int465 /*\n\n398 int466 /*\n\n399 int467 /*\n\n400 int468 /*\n\n401 int469 /*\n\n402 int470 /*\n\n403 int471 /*\n\n404 int472 /*\n\n405 int473 /*\n\n406 int474 /*\n\n407 int475 /*\n\n408 int476 /*\n\n409 int477 /*\n\n410 int478 /*\n\n411 int479 /*\n\n412 int480 /*\n\n413 int481 /*\n\n414 int482 /*\n\n415 int483 /*\n\n416 int484 /*\n\n417 int485 /*\n\n418 int486 /*\n\n419 int487 /*\n\n420 int488 /*\n\n421 int489 /*\n\n422 int490 /*\n\n423 int491 /*\n\n424 int492 /*\n\n425 int493 /*\n\n426 int494 /*\n\n427 int495 /*\n\n428 int496 /*\n\n429 int497 /*\n\n430 int498 /*\n\n431 int499 /*\n\n432 int500 /*\n\n433 int501 /*\n\n434 int502 /*\n\n435 int503 /*\n\n436 int504 /*\n\n437 int505 /*\n\n438 int506 /*\n\n439 int507 /*\n\n440 int508 /*\n\n441 int509 /*\n\n442 int510 /*\n\n443 int511 /*\n\n444 int512 /*\n\n445 int513 /*\n\n446 int514 /*\n\n447 int515 /*\n\n448 int516 /*\n\n449 int517 /*\n\n450 int518 /*\n\n451 int519 /*\n\n452 int520 /*\n\n453 int521 /*\n\n454 int522 /*\n\n455 int523 /*\n\n456 int524 /*\n\n457 int525 /*\n\n458 int526 /*\n\n459 int527 /*\n\n460 int528 /*\n\n461 int529 /*\n\n462 int530 /*\n\n463 int531 /*\n\n464 int532 /*\n\n465 int533 /*\n\n466 int534 /*\n\n467 int535 /*\n\n468 int536 /*\n\n469 int537 /*\n\n470 int538 /*\n\n471 int539 /*\n\n472 int540 /*\n\n473 int541 /*\n\n474 int542 /*\n\n475 int543 /*\n\n476 int544 /*\n\n477 int545 /*\n\n478 int546 /*\n\n479 int547 /*\n\n480 int548 /*\n\n481 int549 /*\n\n482 int550 /*\n\n483 int551 /*\n\n484 int552 /*\n\n485 int553 /*\n\n486 int554 /*\n\n487 int555 /*\n\n488 int556 /*\n\n489 int557 /*\n\n490 int558 /*\n\n491 int559 /*\n\n492 int560 /*\n\n493 int561 /*\n\n494 int562 /*\n\n495 int563 /*\n\n496 int564 /*\n\n497 int565 /*\n\n498 int566 /*\n\n499 int567 /*\n\n500 int568 /*\n\n501 int569 /*\n\n502 int570 /*\n\n503 int571 /*\n\n504 int572 /*\n\n505 int573 /*\n\n506 int574 /*\n\n507 int575 /*\n\n508 int576 /*\n\n509 int577 /*\n\n510 int578 /*\n\n511 int579 /*\n\n512 int580 /*\n\n513 int581 /*\n\n514 int582 /*\n\n515 int583 /*\n\n516 int584 /*\n\n517 int585 /*\n\n518 int586 /*\n\n519 int587 /*\n\n520 int588 /*\n\n521 int589 /*\n\n522 int590 /*\n\n523 int591 /*\n\n524 int592 /*\n\n525 int593 /*\n\n526 int594 /*\n\n527 int595 /*\n\n528 int596 /*\n\n529 int597 /*\n\n530 int598 /*\n\n531 int599 /*\n\n532 int600 /*\n\n533 int601 /*\n\n534 int602 /*\n\n535 int603 /*\n\n536 int604 /*\n\n537 int605 /*\n\n538 int606 /*\n\n539 int607 /*\n\n540 int608 /*\n\n541 int609 /*\n\n542 int610 /*\n\n543 int611 /*\n\n544 int612 /*\n\n545 int613 /*\n\n546 int614 /*\n\n547 int615 /*\n\n548 int616 /*\n\n549 int617 /*\n\n550 int618 /*\n\n551 int619 /*\n\n552 int620 /*\n\n553 int621 /*\n\n554 int622 /*\n\n555 int623 /*\n\n556 int624 /*\n\n557 int625 /*\n\n558 int626 /*\n\n559 int627 /*\n\n560 int628 /*\n\n561 int629 /*\n\n562 int630 /*\n\n563 int631 /*\n\n564 int632 /*\n\n565 int633 /*\n\n566 int634 /*\n\n567 int635 /*\n\n568 int636 /*\n\n569 int637 /*\n\n570 int638 /*\n\n571 int639 /*\n\n572 int640 /*\n\n573 int641 /*\n\n574 int642 /*\n\n575 int643 /*\n\n576 int644 /*\n\n577 int645 /*\n\n578 int646 /*\n\n579 int647 /*\n\n580 int648 /*\n\n581 int649 /*\n\n582 int650 /*\n\n583 int651 /*\n\n584 int652 /*\n\n585 int653 /*\n\n586 int654 /*\n\n587 int655 /*\n\n588 int656 /*\n\n589 int657 /*\n\n590 int658 /*\n\n591 int659 /*\n\n592 int660 /*\n\n593 int661 /*\n\n594 int662 /*\n\n595 int663 /*\n\n596 int664 /*\n\n597 int665 /*\n\n598 int666 /*\n\n599 int667 /*\n\n600 int668 /*\n\n601 int669 /*\n\n602 int670 /*\n\n603 int671 /*\n\n604 int672 /*\n\n605 int673 /*\n\n606 int674 /*\n\n607 int675 /*\n\n608 int676 /*\n\n609 int677 /*\n\n610 int678 /*\n\n611 int679 /*\n\n612 int680 /*\n\n613 int681 /*\n\n614 int682 /*\n\n615 int683 /*\n\n616 int684 /*\n\n617 int685 /*\n\n618 int686 /*\n\n619 int687 /*\n\n620 int688 /*\n\n621 int689 /*\n\n622 int690 /*\n\n623 int691 /*\n\n624 int692 /*\n\n625 int693 /*\n\n626 int694 /*\n\n627 int695 /*\n\n628 int696 /*\n\n629 int697 /*\n\n630 int698 /*\n\n631 int699 /*\n\n632 int700 /*\n\n633 int701 /*\n\n634 int702 /*\n\n635 int703 /*\n\n636 int704 /*\n\n637 int705 /*\n\n638 int706 /*\n\n639 int707 /*\n\n640 int708 /*\n\n641 int709 /*\n\n642 int710 /*\n\n643 int711 /*\n\n644 int712 /*\n\n645 int713 /*\n\n646 int714 /*\n\n647 int715 /*\n\n648 int716 /*\n\n649 int717 /*\n\n650 int718 /*\n\n651 int719 /*\n\n652 int720 /*\n\n653 int721 /*\n\n654 int722 /*\n\n655 int723 /*\n\n656 int724 /*\n\n657 int725 /*\n\n658 int726 /*\n\n659 int727 /*\n\n660 int728 /*\n\n661 int729 /*\n\n662 int730 /*\n\n663 int731 /*\n\n664 int732 /*\n\n665 int733 /*\n\n666 int734 /*\n\n667 int735 /*\n\n668 int736 /*\n\n669 int737 /*\n\n670 int738 /*\n\n671 int739 /*\n\n672 int740 /*\n\n673 int741 /*\n\n674 int742 /*\n\n675 int743 /*\n\n676 int744 /*\n\n677 int745 /*\n\n678 int746 /*\n\n679 int747 /*\n\n680 int748 /*\n\n681 int749 /*\n\n682 int750 /*\n\n683 int751 /*\n\n684 int752 /*\n\n685 int753 /*\n\n686 int754 /*\n\n687 int755 /*\n\n688 int756 /*\n\n689 int757 /*\n\n690 int758 /*\n\n691 int759 /*\n\n692 int760 /*\n\n693 int761 /*\n\n694 int762 /*\n\n695 int763 /*\n\n696 int764 /*\n\n697 int765 /*\n\n698 int766 /*\n\n699 int767 /*\n\n700 int768 /*\n\n701 int769 /*\n\n702 int770 /*\n\n703 int771 /*\n\n704 int772 /*\n\n705 int773 /*\n\n706 int774 /*\n\n707 int775 /*\n\n708 int776 /*\n\n709 int777 /*\n\n710 int778 /*\n\n711 int779 /*\n\n712 int780 /*\n\n713 int781 /*\n\n714 int782 /*\n\n715 int783 /*\n\n716 int784 /*\n\n717 int785 /*\n\n718 int786 /*\n\n719 int787 /*\n\n720 int788 /*\n\n721 int789 /*\n\n722 int790 /*\n\n723 int791 /*\n\n724 int792 /*\n\n725 int793 /*\n\n726 int794 /*\n\n727 int795 /*\n\n728 int796 /*\n\n729 int797 /*\n\n730 int798 /*\n\n731 int799 /*\n\n732 int800 /*\n\n733 int801 /*\n\n734 int802 /*\n\n735 int803 /*\n\n736 int804 /*\n\n737 int805 /*\n\n738 int806 /*\n\n739 int807 /*\n\n740 int808 /*\n\n741 int809 /*\n\n742 int810 /*\n\n743 int811 /*\n\n744 int812 /*\n\n745 int813 /*\n\n746 int814 /*\n\n747 int815 /*\n\n748 int816 /*\n\n749 int817 /*\n\n750 int818 /*\n\n751 int819 /*\n\n752 int820 /*\n\n753 int821 /*\n\n754 int822 /*\n\n755 int823 /*\n\n756 int824 /*\n\n757 int825 /*\n\n758 int826 /*\n\n759 int827 /*\n\n760 int828 /*\n\n761 int829 /*\n\n762 int830 /*\n\n763 int831 /*\n\n764 int832 /*\n\n765 int833 /*\n\n766 int834 /*\n\n767 int835 /*\n\n768 int836 /*\n\n769 int837 /*\n\n770 int838 /*\n\n771 int839 /*\n\n772 int840 /*\n\n773 int841 /*\n\n774 int842 /*\n\n775 int843 /*\n\n776 int844 /*\n\n777 int845 /*\n\n778 int846 /*\n\n779 int847 /*\n\n780 int848 /*\n\n781 int849 /*\n\n782 int850 /*\n\n783 int851 /*\n\n784 int852 /*\n\n785 int853 /*\n\n786 int854 /*\n\n787 int855 /*\n\n788 int856 /*\n\n789 int857 /*\n\n790 int858 /*\n\n791 int859 /*\n\n792 int860 /*\n\n793 int861 /*\n\n794 int862 /*\n\n795 int863 /*\n\n796 int864 /*\n\n797 int865 /*\n\n798 int866 /*\n\n799 int867 /*\n\n800 int868 /*\n\n801 int869 /*\n\n802 int870 /*\n\n803 int871 /*\n\n804 int872 /*\n\n805 int873 /*\n\n806 int874 /*\n\n807 int875 /*\n\n808 int876 /*\n\n809 int877 /*\n\n810 int878 /*\n\n811 int879 /*\n\n812 int880 /*\n\n813 int881 /*\n\n814 int882 /*\n\n815 int883 /*\n\n816 int884 /*\n\n817 int885 /*\n\n818 int886 /*\n\n819 int887 /*\n\n820 int888 /*\n\n821 int889 /*\n\n822 int890 /*\n\n823 int891 /*\n\n824 int892 /*\n\n825 int893 /*\n\n826 int894 /*\n\n827 int895 /*\n\n828 int896 /*\n\n829 int897 /*\n\n830 int898 /*\n\n831 int899 /*\n\n832 int900 /*\n\n833 int901 /*\n\n834 int902 /*\n\n835 int903 /*\n\n836 int904 /*\n\n837 int905 /*\n\n838 int906 /*\n\n839 int907 /*\n\n840 int908 /*\n\n841 int909 /*\n\n842 int910 /*\n\n843 int911 /*\n\n844 int912 /*\n\n845 int913 /*\n\n846 int914 /*\n\n847 int915 /*\n\n848 int916 /*\n\n849 int917 /*\n\n850 int918 /*\n\n851 int919 /*\n\n852 int920 /*\n\n853 int921 /*\n\n854 int922 /*\n\n855 int923 /*\n\n856 int924 /*\n\n857 int925 /*\n\n858 int926 /*\n\n859 int927 /*\n\n860 int928 /*\n\n861 int929 /*\n\n862 int930 /*\n\n863 int931 /*\n\n864 int932 /*\n\n865 int933 /*\n\n866 int934 /*\n\n867 int935 /*\n\n868 int936 /*\n\n869 int937 /*\n\n870 int938 /*\n\n871 int939 /*\n\n872 int940 /*\n\n873 int941 /*\n\n874 int942 /*\n\n875 int943 /*\n\n876 int944 /*\n\n877 int945 /*\n\n878 int946 /*\n\n879 int947 /*\n\n880 int948 /*\n\n881 int949 /*\n\n882 int950 /*\n\n883 int951 /*\n\n884 int952 /*\n\n885 int953 /*\n\n886 int954 /*\n\n887 int955 /*\n\n888 int956 /*\n\n889 int957 /*\n\n890 int958 /*\n\n891 int959 /*\n\n892 int960 /*\n\n893 int961 /*\n\n894 int962 /*\n\n895 int963 /*\n\n896 int964 /*\n\n897 int965 /*\n\n898 int966 /*\n\n899 int967 /*\n\n900 int968 /*\n\n901 int969 /*\n\n902 int970 /*\n\n903 int971 /*\n\n904 int972 /*\n\n905 int973 /*\n\n906 int974 /*\n\n907 int975 /*\n\n908 int976 /*\n\n909 int977 /*\n\n910 int978 /*\n\n911 int979 /*\n\n912 int980 /*\n\n913 int981 /*\n\n914 int982 /*\n\n915 int983 /*\n\n916 int984 /*\n\n917 int985 /*\n\n918 int986 /*\n\n919 int987 /*\n\n920 int988 /*\n\n921 int989 /*\n\n922 int990 /*\n\n923 int991 /*\n\n924 int992 /*\n\n925 int993 /*\n\n926 int994 /*\n\n927 int995 /*\n\n928 int996 /*\n\n929 int997 /*\n\n930 int998 /*\n\n931 int999 /*\n\n932 int1000 /*\n\n933 int1001 /*\n\n934 int1002 /*\n\n935 int1003 /*\n\n936 int1004 /*\n\n937 int1005 /*\n\n938 int1006 /*\n\n939 int1007 /*\n\n940 int1008 /*\n\n941 int1009 /*\n\n942 int1010 /*\n\n943 int1011 /*\n\n944 int1012 /*\n\n945 int1013 /*\n\n946 int1014 /*\n\n947 int1015 /*\n\n948 int1016 /*\n\n949 int1017 /*\n\n950 int1018 /*\n\n951 int1019 /*\n\n952 int1020 /*\n\n953 int1021 /*\n\n954 int1022 /*\n\n955 int1023 /*\n\n956 int1024 /*\n\n957 int1025 /*\n\n958 int1026 /*\n\n959 int1027 /*\n\n960 int1028 /*\n\n961 int1029 /*\n\n962 int1030 /*\n\n963 int1031 /*\n\n964 int1032 /*\n\n965 int1033 /*\n\n966 int1034 /*\n\n967 int1035 /*\n\n968 int1036 /*\n\n969 int1037 /*\n\n970 int1038 /*\n\n971 int1039 /*\n\n972 int1040 /*\n\n973 int1041 /*\n\n974 int1042 /*\n\n975 int1043 /*\n\n976 int1044 /*\n\n977 int1045 /*\n\n978 int1046 /*\n\n979 int1047 /*\n\n980 int1048 /*\n\n981 int1049 /*\n\n982 int1050 /*\n\n983 int1051 /*\n\n984 int1052 /*\n\n985 int1053 /*\n\n986 int1054 /*\n\n987 int1055 /*\n\n988 int1056 /*\n\n989 int1057 /*\n\n990 int1058 /*\n\n991 int1059 /*\n\n992 int1060 /*\n\n993 int1061 /*\n\n994 int1062 /*\n\n995 int1063 /*\n\n996 int1064 /*\n\n997 int1065 /*\n\n998 int1066 /*\n\n999 int1067 /*\n\n1000 int1068 /*\n\n1001 int1069 /*\n\n1002 int1070 /*\n\n1003 int1071 /*\n\n1004 int1072 /*\n\n1005 int1073 /*\n\n1006 int1074 /*\n\n1007 int1075 /*\n\n1008 int1076 /*\n\n1009 int1077 /*\n\n1010 int1078 /*\n\n1011 int1079 /*\n\n1012 int1080 /*\n\n1013 int1081 /*\n\n1014 int1082 /*\n\n1015 int1083 /*\n\n1016 int1084 /*\n\n1017 int1085 /*\n\n1018 int1086 /*\n\n1019 int1087 /*\n\n1020 int1088 /*\n\n1021 int1089 /*\n\n1022 int1090 /*\n\n1023 int1091 /*\n\n1024 int1092 /*\n\n1025 int1093 /*\n\n1026 int1094 /*\n\n1027 int1095 /*\n\n1028 int1096 /*\n\n1029 int1097 /*\n\n1030 int1098 /*\n\n1031 int1099 /*\n\n1032 int1100 /*\n\n1033 int1101 /*\n\n1034 int1102 /*\n\n1035 int1103 /*\n\n1036 int1104 /*\n\n1037 int1105 /*\n\n1038 int1106 /*\n\n1039 int1107 /*\n\n1040 int1108 /*\n\n1041 int1109 /*\n\n1042 int1110 /*\n\n1043 int1111 /*\n\n1044 int1112 /*\n\n1045 int1113 /*\n\n1046 int1114 /*\n\n1047 int1115 /*\n\n1048 int1116 /*\n\n1049 int1117 /*\n\n1050 int1118 /*\n\n1051 int1119 /*\n\n1052 int1120 /*\n\n1053 int1121 /*\n\n1054 int1122 /*\n\n1055 int1123 /*\n\n1056 int1124 /*\n\n1057 int1125 /*\n\n1058 int1126 /*\n\n1059 int1127 /*\n\n1060 int1128 /*\n\n1061 int1129 /*\n\n1062 int1130 /*\n\n1063 int1131 /*\n\n1064 int1132 /*\n\n1065 int1133 /*\n\n1066 int1134 /*\n\n1067 int1135 /*\n\n1068 int1136 /*\n\n1069 int1137 /*\n\n1070 int1138 /*\n\n1071 int1139 /*\n\n1072 int1140 /*\n\n1073 int1141 /*\n\n1074 int1142 /*\n\n1075 int1143 /*\n\n1076 int1144 /*\n\n1077 int1145 /*\n\n1078 int1146 /*\n\n1079 int1147 /*\n\n1080 int1148 /*\n\n1081 int1149 /*\n\n1082 int1150 /*\n\n1083 int1151 /*\n\n1084 int1152 /*\n\n1085 int1153 /*\n\n1086 int1154 /*\n\n1087 int1155 /*\n\n1088 int1156 /*\n\n1089 int1157 /*\n\n1090 int1158 /*\n\n1091 int1159 /*\n\n1092 int1160 /*\n\n1093 int1161 /*\n\n1094 int1162 /*\n\n1095 int1163 /*\n\n1096 int1164 /*\n\n1097 int1165 /*\n\n1098 int1166 /*\n\n1099 int1167 /*\n\n1100 int1168 /*\n\n1101 int1169 /*\n\n1102 int1170 /*\n\n1103 int1171 /*\n\n1104 int1172 /*\n\n1105 int1173 /*\n\n1106 int1174 /*\n\n1107 int1175 /*\n\n1108 int1176 /*\n\n1109 int1177 /*\n\n1110 int1178 /*\n\n1111 int1179 /*\n\n1112 int1180 /*\n\n1113 int1181 /*\n\n1114 int1182 /*\n\n1115 int1183 /*\n\n1116 int1184 /*\n\n1117 int1185 /*\n\n1118 int1186 /*\n\n1119 int1187 /*\n\n1120 int1188 /*\n\n1121 int1189 /*\n\n1122 int1190 /*\n\n1123 int1191 /*\n\n1124 int1192 /*\n\n1125 int1193 /*\n\n1126 int1194 /*\n\n1127 int1195 /*\n\n1128 int1196 /*\n\n1129 int1197 /*\n\n1130 int1198 /*\n\n1131 int1199 /*\n\n1132 int1200 /*\n\n1133 int1201 /*\n\n1134 int1202 /*\n\n1135 int1203 /*\n\n1136 int1204 /*\n\n1137 int1205 /*\n\n1138 int1206 /*\n\n1139 int1207 /*\n\n1140 int1208 /*\n\n1141 int1209 /*\n\n1142 int1210 /*\n\n1143 int1211 /*\n\n1144 int1212 /*\n\n1145 int1213 /*\n\n1146 int1214 /*\n\n1147 int1215 /*\n\n1148 int1216 /*\n\n1149 int1217 /*\n\n1150 int1218 /*\n\n1151 int1219 /*\n\n1152 int1220 /*\n\n1153 int1221 /*\n\n1154 int1222 /*\n\n1155 int1223 /*\n\n1156 int1224 /*\n\n1157 int1225 /*\n\n1158 int1226 /*\n\n1159 int1227 /*\n\n1160 int1228 /*\n\n1161 int1229 /*\n\n1162 int1230 /*\n\n1163 int1231 /*\n\n1164 int1232 /*\n\n1165 int1233 /*\n\n1166 int1234 /*\n\n1167 int1235 /*\n\n1168 int1236 /*\n\n1169 int1237 /*\n\n1170 int1238 /*\n\n1171 int1239 /*\n\n1172 int1240 /*\n\n1173 int1241 /*\n\n1174 int1242 /*\n\n1175 int1243 /*\n\n1176 int1244 /*\n\n1177 int1245 /*\n\n1178 int1246 /*\n\n1179 int1247 /*\n\n1180 int1248 /*\n\n1181 int1249 /*\n\n1182 int1250 /*\n\n1183 int1251 /*\n\n1184 int1252 /*\n\n1185 int1253 /*\n\n1186 int1254 /*\n\n1187 int1255 /*\n\n1188 int1256 /*\n\n1189 int1257 /*\n\n1190 int1258 /*\n\n1191 int1259 /*\n\n1192 int1260 /*\n\n1193 int1261 /*\n\n1194 int1262 /*\n\n1195 int1263 /*\n\n1196 int1264 /*\n\n1197 int1265 /*\n\n1198 int1266 /*\n\n1199 int1267 /*\n\n1200 int1268 /*\n\n1201 int1269 /*\n\n1202 int1270 /*\n\n1203 int1271 /*\n\n1204 int1272 /*\n\n1205 int1273 /*\n\n1206 int1274 /*\n\n1207 int1275 /*\n\n1208 int1276 /*\n\n1209 int12
```





```
std::allocator<signed char> > const&,
arrow::UnionMode::Type>:: compressed_pair<std::allocator<arrow::UnionTypes &, const
std::vector<std::shared_ptr<arrow::Field> > >, const std::allocator<signed char,
std::allocator<std::shared_ptr<arrow::Field> > >, const std::allocator<signed char,
UnimplementedBehaviorSanitizer: undefined-behavior /usr/local/include/++/vector:1549:12 in',
binary_path": "/out/arrow-ipc-stream-fuzz", "cid": "996c9d6b47c"}))
```

**Thought**

Planner call: run\_gdb

```
$ run_gdb(binary_path"/out/arrow-ipc-stream-fuzz")
```

**Thought**

Planner result: run\_gdb

**Observation**

```
{ "ok": true, "warning": "Failed to install gdb inside the container." }
```

**Thought**

Planner call: delegate

```
$ delegate(task="llm_analyze() - extract_json() and
publish_verified_locations(repo, root="/src/libreoffice"). If available,
compare_llm_metrics(diff_index_json_path="/diff_index.json",
llm_json_path="/published_json", chunks_json_path="/case_chunks.json",
line_slack=0), Finally, finish_task(summary="include metrics + paths");)
```

**Thought**

Planner result: delegate

**Observation**

```
"no improvement in 3 rounds, end executor"
```

**Thought**

Planner call: giveup

```
$ giveup(confirm=true)
```

**Thought**

Planner result: giveup

**Observation**

```
{ "success": true }
```

**Thought**

Executor#0 note

**Observation**

You are the EXECUTOR for ARVO bug localization. You run heavy tools and MUST call finish\_task(summary=...) to return results.

**TOOLS**

- llm\_analyze(crash\_log=<string> refinement\_bundle=..., top\_n=<int>)
- extract\_json(response\_text, merge\_with\_last=<bool>)
- view\_source(file\_path, start\_line=None, end\_line=None, max\_bytes=200000)
- grep\_source(pattern, root="/src", max\_results=200)
- copy\_out(path, out\_path=None)
- compare\_llm\_metrics(diff\_index\_json\_path, llm\_json\_path=None, chunks\_json\_path=None, line\_slack=0)
- finish\_task(summary=<string>)

**RUNTIME OWNERSHIP**

- Do NOT run ASAN or GDB. The Planner owns runtime collection and has cached 'crash log'.
- You MAY call llm\_analyze() with NO arguments to consume the cached crash log from the environment.

**REFINEMENT** (argument-only, optional):

- Use view\_source/grep\_source to fetch code slices to promising files/functions.
- Then call llm\_analyze/refinement\_bundle() to GENERATE A NEW batch of candidates from these slices only.
- This refinement MUST NOT narrow, edit, or reuse previous candidates; it produces up to exactly <M> fresh items.
- After refinement, call extract\_json(response\_text, merge\_with\_last=false) so the new batch stays separate from the first batch.

**COMPARISON**

- You may call compare\_llm\_metrics multiple times. Evaluate the first JSON snapshot, optionally refine to produce a new batch; then evaluate again.
- Provide the correct llm\_json\_path for each compare; do not call compare\_llm\_metrics without it.

**NO-OP/INVALID CALLS**

- Do NOT re-run llm\_analyze repeatedly for the same inputs (at most one argument round).
- Do NOT open files >200KB; keep total view\_source =<5 and grep\_source =<2.

**FINISH CONTRACT:**

- finish\_task(summary=...) must be a single-line, semicolon-separated key=val list including: ok=<true/false>, detection\_rate=<float>, localization\_rate=<float>, strict\_localization\_rate=<float>, improved=<true/false>, prev=<float>, curr=<float>.
- diff\_index\_json\_path=<path>, chunks\_json\_path=<path>, llm\_json\_path=<path or none>.
- gdb=<unknown>, bin=<unknown>.

**Thought**

Executor#0 note

**Observation**

You have been delegated a task:  
Use the cached crash log produced by the Planner. Analyze, optionally augment with source views, evaluate twice, then return metrics.

**Steps:**

- llm\_analyze() # no args -> consume cached crash log
- extract\_json(response\_text, merge\_with\_last=false) # first JSON snapshot (batch A)
- compare\_llm\_metrics(diff\_index\_json\_path="/diff\_index.json", llm\_json\_path="/batch\_A\_JSON", chunks\_json\_path="/case\_chunks.json", line\_slack=0)
- SOURCE PEEK (optional, before second compare):
  - Parse crash log for file paths (e.g., "/src/...<lines>") and function names.
  - Include from 1-2 snippets from batch A. Exclude on crash state from Batch B/prev.

For up to TOP\_K candidates (decide by file):  
- If file\_path known: view\_source(file\_path, start\_line=40, end\_line=40, max\_bytes=200000)  
- Else: grep\_source(symbol/function, root="/src", max\_results=20) and view\_source best match <=40  
Refines: <5 view source and <2 grep\_source; skip files >200KB

5) REFINEMENT (argument-only, one pass):  
# Generates a NEW batch (batch B) from slices only; does not modify batch A  
# extract\_json(response\_text, merge\_with\_last=false) # keep batch B separate  
6) compare\_llm\_metrics(diff\_index\_json\_path="/diff\_index.json", llm\_json\_path="/batch\_B\_JSON", chunks\_json\_path="/case\_chunks.json", line\_slack=0)  
7) finish\_task(summary="ok=<...>; detection\_rate=<...>; localization\_rate=<...>; strict\_localization\_rate=<...>; improved=<...>; prev=<...>; curr=<...>")  
diff\_index\_json\_path="/diff\_index.json", chunks\_json\_path="/case\_chunks.json", llm\_json\_path="/batch\_B\_JSON", gdb=<unknown>, bin=<unknown>)

**Thought**

Executor#0 call: llm\_analyze

```
$ llm_analyze()
```

**Thought**

Executor#0 result: llm\_analyze

**Observation**

```
{ "ok": true, "mode": "crash", "text": "\n\n({ \"file_path\": \"/src/arrow/cpp/src/arrow/type.cc\", \"start_line\": 362, \"end_line\": 366, \"confidence\": \"very likely\", \"reason\": \"Constructor uses vector access which may lead to out-of-bounds access, linked to the crash reported on operator[]\", \"symbol\": \"arrow::UnionType::UnionType\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/type.cc\", \"start_line\": 1397, \"end_line\": 1412, \"confidence\": \"likely\", \"reason\": \"Potential unsafe vector access in union_function can cause similar issues as reported.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/metadata_internal.cc\", \"start_line\": 175, \"end_line\": 186, \"confidence\": \"possible\", \"reason\": \"FieldFromFlatbuffer may not correctly check boundaries for field types leading to crashes.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/metadata_internal.cc\", \"start_line\": 358, \"end_line\": 370, \"confidence\": \"possible\", \"reason\": \"Similar potential for unsafe access in handling of UnionType in Flatbuffer.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/metadata_internal.cc\", \"start_line\": 378, \"end_line\": 378, \"confidence\": \"possible\", \"reason\": \"FieldFromFlatbuffer may not correctly check boundaries for field types leading to crashes.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 446, \"end_line\": 475, \"confidence\": \"unlikely\", \"reason\": \"ReadSchema may depend on the integrity of types being parsed, which can be unsafe.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 585, \"end_line\": 590, \"confidence\": \"unlikely\", \"reason\": \"Unsafe handling of shared ptrs could link to vector issues seen, albeit less direct.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 601, \"end_line\": 610, \"confidence\": \"unlikely\", \"reason\": \"Potential flaws in InputStream handling could lead to mishandled data sizes in vectors.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 25, \"end_line\": 38, \"confidence\": \"very unlikely\", \"reason\": \"Fuzz test may trigger crashes, unlikely related to vector bounds.\", \"in_file_path\": \"/src/lvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp\", \"start_line\": 19, \"end_line\": 24, \"confidence\": \"very unlikely\", \"reason\": \"Main function unlikely to contain root cause of vector index issues seen.\", \"in_file_path\": \"/home/bug_localization_agents/work_arvo/2021/predictions.json\", \"batch_json_path\": \"/home/bug_localization_agents/work_arvo/2021/lm_batch_0001.json\", \"chunks_json_path\": \"/home/bug_localization_agents/work_arvo/2021/cumulative_predictions.json\", \"count\": 10, \"batch\": 1} }
```

**Thought**

Executor#0 call: extract\_json

```
$ extract_json(response_text="{ \"file_path\": \"/src/arrow/cpp/src/arrow/type.cc\", \"start_line\": 362, \"end_line\": 366, \"confidence\": \"very likely\", \"reason\": \"Constructor uses vector access which may lead to out-of-bounds access, linked to the crash reported on operator[]\", \"symbol\": \"arrow::UnionType::UnionType\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/type.cc\", \"start_line\": 1397, \"end_line\": 1412, \"confidence\": \"likely\", \"reason\": \"Potential unsafe vector access in union_function can cause similar issues as reported.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/metadata_internal.cc\", \"start_line\": 175, \"end_line\": 186, \"confidence\": \"possible\", \"reason\": \"FieldFromFlatbuffer may not validate the size correctly leading to crashes as shown.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/metadata_internal.cc\", \"start_line\": 358, \"end_line\": 370, \"confidence\": \"possible\", \"reason\": \"Similar potential for unsafe access in handling of UnionType in Flatbuffer.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/metadata_internal.cc\", \"start_line\": 378, \"end_line\": 378, \"confidence\": \"possible\", \"reason\": \"FieldFromFlatbuffer may not correctly check boundaries for field types leading to crashes.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 446, \"end_line\": 475, \"confidence\": \"unlikely\", \"reason\": \"ReadSchema may depend on the integrity of types being parsed, which can be unsafe.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 585, \"end_line\": 590, \"confidence\": \"unlikely\", \"reason\": \"Unsafe handling of shared ptrs could link to vector issues seen, albeit less direct.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 601, \"end_line\": 610, \"confidence\": \"unlikely\", \"reason\": \"Potential flaws in InputStream handling could lead to mishandled data sizes in vectors.\", \"in_file_path\": \"/src/arrow/cpp/src/arrow/pc/reader.cc\", \"start_line\": 25, \"end_line\": 38, \"confidence\": \"very unlikely\", \"reason\": \"Fuzz test may trigger crashes, unlikely related to vector bounds.\", \"in_file_path\": \"/src/lvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp\", \"start_line\": 19, \"end_line\": 24, \"confidence\": \"very unlikely\", \"reason\": \"Main function unlikely to contain root cause of vector index issues seen.\", \"symbol\": \"\", \"in_file_path\": \"/home/bug_localization_agents/work_arvo/2021/predictions.json\", \"batch_json_path\": \"/home/bug_localization_agents/work_arvo/2021/lm_batch_0001.json\", \"chunks_json_path\": \"/home/bug_localization_agents/work_arvo/2021/cumulative_predictions.json\", \"count\": 10, \"batch\": 1, \"merge_with_last\": false} ")
```

**Thought**

Executor#0 result: extract\_json

**Observation**

```
{ "ok": true, "json": { \"llm_predictions\": [ { \"file_path\": \"/src/arrow/cpp/src/arrow/type.cc\", \"start_line\": 362, \"end_line\": 366, \"confidence\": \"very likely\", \"reason\": \"Constructor uses vector access which may
```





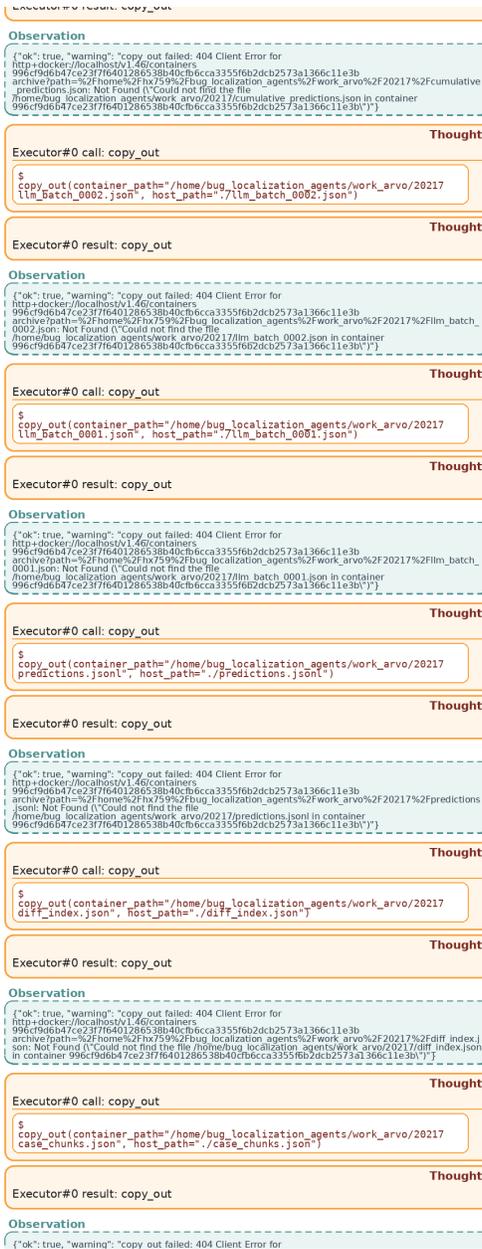


Figure 16: GPT-4o mini Divergence Tracing for case 20217.

```

Prompt
Task: Orchestrate ARVO bug localization and evaluation.
Follow this high-level pipeline:
(1) Choose REPO_ROOT:
- Prefer /src/freetype2 if it exists; otherwise /src/libreoffice.
(2) chunk_case(root_dir=REPO_ROOT, out_json_path="/case_chunks.json")
(3) diff_index(diff_path="/home/bug_localization_agents/work_arvo/25357/diff_index.json")
(4) mark_diff(diff_json_path="/diff_index.json", chunks_json_path="/case_chunks.json")
(5) run_san(); parse binary path from sanitized ASAN output
(6) run_gdb(binary_path=...) ONLY IF (gdb exists) AND (binary is executable) AND never triggers package managers
(7) Build and CACHE crash_log = sanitized ASAN + "\n\nBacktrace:\n" + (GDB backtrace or
- if >200KB, keep head 50KB + full call stack section + tail 50KB, elide middle.
(8) DELEGATE to the Executor. The delegated task MUST ask for:
- ltm_analyze(crash_log=...) - extract_json(response_text)
- Prefer calling ltm_analyze() with no args to consume the cached crash_log from Planner.
- Optionally: view_source / grep_source around high-confidence lines -> build_refinement_bundle
- ltm_analyze(refinement_bundle=...) - extract_json(ONE_PASS)
- If available: publish_verified_locations(..., repo_root=REPO_ROOT)
- compare_ltm_metrics(diff_index_json_path="/diff_index.json", ltm_json_path=verified_or_raw_JSON, chunks_json_path="/case_chunks.json", line_stack=0)
- Finally: finish_task(summary=<<include metrics + paths>>)
Output only tool calls or delegated tasks as needed. Keep text minimal.

```

**Thought**

**Planner note**

```

Observation
You are the PLANNER for ARVO bug localization. You orchestrate the pipeline, call "lightweight reasoning" tools and runtime collection (ASAN/GDB) yourself, and DELEGATE heavy reasoning/evaluation to the Executor via delegate(task=...).

## Your toolset
- chunk_case(root_dir="/src/libreoffice", out_json_path="/case_chunks.json", include_context=0, max_files=100000)
- diff_index(diff_path="/diff_index.json")
- mark_diff(diff_json_path="/diff_index.json", chunks_json_path="/case_chunks.json")
- run_san()
- run_gdb(binary_path="/vuln")
- delegate(task=<string>)
- giveup()

## Responsibilities
- OWN orchestration and state: start container once, choose REPO_ROOT, build chunks & diff index once, mark_diff chunks once (unless inputs changed).
- YOU run ASAN/GDB and cache crash_log in environment state.
- For LTM reasoning & evaluation: DELEGATE to the Executor with a clear, structured task that includes:
  - What to analyze (use cached crash_log -> ltm_analyze -> extract_json).
  - Which files to use (paths to diff_index.json, case_chunks.json).
  - Whether to verify (publish_verified_locations if available) -> compare_ltm_metrics.
  - Output contract (return metrics and paths in "finish_task(summary=...)", if appropriate results. -> Do not call giveup!) after a single delegation. -> Prefer at least 2-3 refinement rounds unless clearly hopeless.

## Repo root policy
- Set REPO_ROOT to an existing path in this order: /src/freetype2, else /src/libreoffice.
- Use that REPO_ROOT consistently when asking the Executor to verify locations.

## Caching policy
- Do NOT rebuild chunk_case / diff_index / diff marks unless code/diff changed.
- Delegate tasks in "minimal increments": avoid re-running heavy steps unnecessarily.

```

**Thought**

**Planner note**

```

Observation
Task: Orchestrate ARVO bug localization and evaluation.
Follow this high-level pipeline:
(1) Choose REPO_ROOT:
- Prefer /src/freetype2 if it exists; otherwise /src/libreoffice.
(2) chunk_case(root_dir=REPO_ROOT, out_json_path="/case_chunks.json")
(3) diff_index(diff_path="/home/bug_localization_agents/work_arvo/25357/diff_index.json")
(4) mark_diff(diff_json_path="/diff_index.json", chunks_json_path="/case_chunks.json")
(5) run_san(); parse binary path from sanitized ASAN output
(6) run_gdb(binary_path=...) ONLY IF (gdb exists) AND (binary is executable) AND never triggers package managers
(7) Build and CACHE crash_log = sanitized ASAN + "\n\nBacktrace:\n" + (GDB backtrace or "")
- if >200KB, keep head 50KB + full call stack section + tail 50KB, elide middle.
(8) DELEGATE to the Executor. The delegated task MUST ask for:
- ltm_analyze(crash_log=...) -> extract_json(response_text)
- Prefer calling ltm_analyze() with no args to consume the cached crash_log from Planner.
- Optionally: view_source / grep_source around high-confidence lines -> build_refinement_bundle
- ltm_analyze(refinement_bundle=...) -> extract_json(ONE_PASS)
- If available: publish_verified_locations(..., repo_root=REPO_ROOT)
- compare_ltm_metrics(diff_index_json_path="/diff_index.json", ltm_json_path=verified_or_raw_JSON, chunks_json_path="/case_chunks.json", line_stack=0)
- Finally: finish_task(summary=<<include metrics + paths>>)
Output only tool calls or delegated tasks as needed. Keep text minimal.

```

**Thought**

**Planner call: chunk\_case**

```

$ chunk_case(root_dir="/src/libreoffice", out_json_path="/home/bug_localization_agents/work_arvo/25357/case_chunks.json")

```

**Thought**

**Planner result: chunk\_case**

**Observation**

```

{"ok": true, "root_dir": "/src/libreoffice", "json_path": "/home/bug_localization_agents/work_arvo/25357/case_chunks.json", "num_files": 362, "num_chunks": 2882}

```

**Thought**

**Planner call: diff\_index**

```

$ diff_index(diff_path="/home/bug_localization_agents/ARVO-Meta/patches/25357_diff_index.json", out_json_path="/home/bug_localization_agents/work_arvo/25357/diff_index.json")

```

**Thought**

**Planner result: diff\_index**

**Observation**

```

{"ok": true, "json_path": "/home/bug_localization_agents/work_arvo/25357/diff_index.json", "num_files": 1, "num_chunks": 0, "num_per_line": 24}

```

**Thought**

**Planner call: mark\_diff**

```

$ mark_diff(diff_json_path="/home/bug_localization_agents/work_arvo/25357/diff_index.json", chunks_json_path="/home/bug_localization_agents/work_arvo/25357/case_chunks.json")

```

**Thought**

**Planner result: mark\_diff**

**Observation**

```

{"ok": true, "chunks_json_path": "/home/bug_localization_agents/work_arvo/25357/case_chunks.json", "num_chunks": 2982, "num_chunks_with_diff": 3}

```

**Thought**

**Planner call: run\_san**

```

$ run_san()

```

**Thought**

**Planner result: run\_san**

**Observation**

```

{"ok": true, "asan_output": {"ok": true, "asan_output": {"INFO: Seed: 4214768504\nINFO: Loaded 1 modules (9995 inline 8-bit counters): 9995 (0x9283c0, 0x9282ac),\n\nINFO: Loaded 1 PC tables (9995 PCs): 9995 (0x927a0, 0x9b9850).\n\nVulnOut:\n\nfuzzer: Running 1 inputs + 1 time(s) each.\n\nRunning:\n\nmpc\n\n=====\n\nvuln001b1\n\nvuln001b131m==2589==ERROR: AddressSanitizer: negative-size-param (size=-539033444)\n\nvuln001b1\n\nvuln001b1mcs\n\nCARNISS: 10 (negative-size-param)\n\n#0 0x5201a4 in __asan_memcpy /src/vuln-project/compiler-rt/lib/asan/asan_interceptors/memintinsics.cpp:22:3\n\n#1 0x5530e0 in memcpy /usr/include/x86_64-linux-gnu/libc.so.2:53:10\n\n#2 0x5530e0 in cf_read_header /src/libstdc++/src/common-c89:37:2\n\n#3 0x547670 in psf_binheader_read /src/libstdc++/src/common-c98:20\n\n#4 0x577637 in cf_read_header /src/libstdc++/src/common-c98:37:2\n\n#5 0x54e62 in psf_open /src/libstdc++/src/sndfile.c:3180:13\n\n#6 0x555c27 in sf_open_virtual /src/libstdc++/src/sndfile.c:440:9\n\n#7 0x553576 in LVW_FuzzerTestOnInput /src/libstdc++/src/sndfile_fuzzer.cc:99:13\n\n#8 0x5458a4 in fuzzer-Fuzzer:ExecuteCallback(unsigned char const*, unsigned long) /src/vuln-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:559:15\n\n#9 0x4441e2 in fuzzer-Fuzzer:RunOneTest(Fuzzer::Char***, int*, unsigned long) /src/vuln-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:301:6\n\n#10 0x44a17d in fuzzer-Fuzzer:RunDriver(char***, int*, unsigned long) /src/vuln-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:801:9\n\n#11 0x471b62 in main /lib64/libc.so.2:2083\n\n#12 0x41e988 in start /lib64/libc.so.2:2083\n\n#13 0x41e988 in start /lib64/libc.so.2:2083\n\n#14 0x41e988 in start /lib64/libc.so.2:2083\n\n#15 0x41e988 in start /lib64/libc.so.2:2083\n\n#16 0x41e988 in start /lib64/libc.so.2:2083\n\n#17 0x41e988 in start /lib64/libc.so.2:2083\n\n#18 0x41e988 in start /lib64/libc.so.2:2083\n\n#19 0x41e988 in start /lib64/libc.so.2:2083\n\n#20 0x41e988 in start /lib64/libc.so.2:2083\n\n#21 0x41e988 in start /lib64/libc.so.2:2083\n\n#22 0x41e988 in start /lib64/libc.so.2:2083\n\n#23 0x41e988 in start /lib64/libc.so.2:2083\n\n#24 0x41e988 in start /lib64/libc.so.2:2083\n\n#25 0x41e988 in start /lib64/libc.so.2:2083\n\n#26 0x41e988 in start /lib64/libc.so.2:2083\n\n#27 0x41e988 in start /lib64/libc.so.2:2083\n\n#28 0x41e988 in start /lib64/libc.so.2:2083\n\n#29 0x41e988 in start /lib64/libc.so.2:2083\n\n#30 0x41e988 in start /lib64/libc.so.2:2083\n\n#31 0x41e988 in start /lib64/libc.so.2:2083\n\n#32 0x41e988 in start /lib64/libc.so.2:2083\n\n#33 0x41e988 in start /lib64/libc.so.2:2083\n\n#34 0x41e988 in start /lib64/libc.so.2:2083\n\n#35 0x41e988 in start /lib64/libc.so.2:2083\n\n#36 0x41e988 in start /lib64/libc.so.2:2083\n\n#37 0x41e988 in start /lib64/libc.so.2:2083\n\n#38 0x41e988 in start /lib64/libc.so.2:2083\n\n#39 0x41e988 in start /lib64/libc.so.2:2083\n\n#40 0x41e988 in start /lib64/libc.so.2:2083\n\n#41 0x41e988 in start /lib64/libc.so.2:2083\n\n#42 0x41e988 in start /lib64/libc.so.2:2083\n\n#43 0x41e988 in start /lib64/libc.so.2:2083\n\n#44 0x41e988 in start /lib64/libc.so.2:2083\n\n#45 0x41e988 in start /lib64/libc.so.2:2083\n\n#46 0x41e988 in start /lib64/libc.so.2:2083\n\n#47 0x41e988 in start /lib64/libc.so.2:2083\n\n#48 0x41e988 in start /lib64/libc.so.2:2083\n\n#49 0x41e988 in start /lib64/libc.so.2:2083\n\n#50 0x41e988 in start /lib64/libc.so.2:2083\n\n#51 0x41e988 in start /lib64/libc.so.2:2083\n\n#52 0x41e988 in start /lib64/libc.so.2:2083\n\n#53 0x41e988 in start /lib64/libc.so.2:2083\n\n#54 0x41e988 in start /lib64/libc.so.2:2083\n\n#55 0x41e988 in start /lib64/libc.so.2:2083\n\n#56 0x41e988 in start /lib64/libc.so.2:2083\n\n#57 0x41e988 in start /lib64/libc.so.2:2083\n\n#58 0x41e988 in start /lib64/libc.so.2:2083\n\n#59 0x41e988 in start /lib64/libc.so.2:2083\n\n#60 0x41e988 in start /lib64/libc.so.2:2083\n\n#61 0x41e988 in start /lib64/libc.so.2:2083\n\n#62 0x41e988 in start /lib64/libc.so.2:2083\n\n#63 0x41e988 in start /lib64/libc.so.2:2083\n\n#64 0x41e988 in start /lib64/libc.so.2:2083\n\n#65 0x41e988 in start /lib64/libc.so.2:2083\n\n#66 0x41e988 in start /lib64/libc.so.2:2083\n\n#67 0x41e988 in start /lib64/libc.so.2:2083\n\n#68 0x41e988 in start /lib64/libc.so.2:2083\n\n#69 0x41e988 in start /lib64/libc.so.2:2083\n\n#70 0x41e988 in start /lib64/libc.so.2:2083\n\n#71 0x41e988 in start /lib64/libc.so.2:2083\n\n#72 0x41e988 in start /lib64/libc.so.2:2083\n\n#73 0x41e988 in start /lib64/libc.so.2:2083\n\n#74 0x41e988 in start /lib64/libc.so.2:2083\n\n#75 0x41e988 in start /lib64/libc.so.2:2083\n\n#76 0x41e988 in start /lib64/libc.so.2:2083\n\n#77 0x41e988 in start /lib64/libc.so.2:2083\n\n#78 0x41e988 in start /lib64/libc.so.2:2083\n\n#79 0x41e988 in start /lib64/libc.so.2:2083\n\n#80 0x41e988 in start /lib64/libc.so.2:2083\n\n#81 0x41e988 in start /lib64/libc.so.2:2083\n\n#82 0x41e988 in start /lib64/libc.so.2:2083\n\n#83 0x41e988 in start /lib64/libc.so.2:2083\n\n#84 0x41e988 in start /lib64/libc.so.2:2083\n\n#85 0x41e988 in start /lib64/libc.so.2:2083\n\n#86 0x41e988 in start /lib64/libc.so.2:2083\n\n#87 0x41e988 in start /lib64/libc.so.2:2083\n\n#88 0x41e988 in start /lib64/libc.so.2:2083\n\n#89 0x41e988 in start /lib64/libc.so.2:2083\n\n#90 0x41e988 in start /lib64/libc.so.2:2083\n\n#91 0x41e988 in start /lib64/libc.so.2:2083\n\n#92 0x41e988 in start /lib64/libc.so.2:2083\n\n#93 0x41e988 in start /lib64/libc.so.2:2083\n\n#94 0x41e988 in start /lib64/libc.so.2:2083\n\n#95 0x41e988 in start /lib64/libc.so.2:2083\n\n#96 0x41e988 in start /lib64/libc.so.2:2083\n\n#97 0x41e988 in start /lib64/libc.so.2:2083\n\n#98 0x41e988 in start /lib64/libc.so.2:2083\n\n#99 0x41e988 in start /lib64/libc.so.2:2083\n\n#100 0x41e988 in start /lib64/libc.so.2:2083\n\n#101 0x41e988 in start /lib64/libc.so.2:2083\n\n#102 0x41e988 in start /lib64/libc.so.2:2083\n\n#103 0x41e988 in start /lib64/libc.so.2:2083\n\n#104 0x41e988 in start /lib64/libc.so.2:2083\n\n#105 0x41e988 in start /lib64/libc.so.2:2083\n\n#106 0x41e988 in start /lib64/libc.so.2:2083\n\n#107 0x41e988 in start /lib64/libc.so.2:2083\n\n#108 0x41e988 in start /lib64/libc.so.2:2083\n\n#109 0x41e988 in start /lib64/libc.so.2:2083\n\n#110 0x41e988 in start /lib64/libc.so.2:2083\n\n#111 0x41e988 in start /lib64/libc.so.2:2083\n\n#112 0x41e988 in start /lib64/libc.so.2:2083\n\n#113 0x41e988 in start /lib64/libc.so.2:2083\n\n#114 0x41e988 in start /lib64/libc.so.2:2083\n\n#115 0x41e988 in start /lib64/libc.so.2:2083\n\n#116 0x41e988 in start /lib64/libc.so.2:2083\n\n#117 0x41e988 in start /lib64/libc.so.2:2083\n\n#118 0x41e988 in start /lib64/libc.so.2:2083\n\n#119 0x41e988 in start /lib64/libc.so.2:2083\n\n#120 0x41e988 in start /lib64/libc.so.2:2083\n\n#121 0x41e988 in start /lib64/libc.so.2:2083\n\n#122 0x41e988 in start /lib64/libc.so.2:2083\n\n#123 0x41e988 in start /lib64/libc.so.2:2083\n\n#124 0x41e988 in start /lib64/libc.so.2:2083\n\n#125 0x41e988 in start /lib64/libc.so.2:2083\n\n#126 0x41e988 in start /lib64/libc.so.2:2083\n\n#127 0x41e988 in start /lib64/libc.so.2:2083\n\n#128 0x41e988 in start /lib64/libc.so.2:2083\n\n#129 0x41e988 in start /lib64/libc.so.2:2083\n\n#130 0x41e988 in start /lib64/libc.so.2:2083\n\n#131 0x41e988 in start /lib64/libc.so.2:2083\n\n#132 0x41e988 in start /lib64/libc.so.2:2083\n\n#133 0x41e988 in start /lib64/libc.so.2:2083\n\n#134 0x41e988 in start /lib64/libc.so.2:2083\n\n#135 0x41e988 in start /lib64/libc.so.2:2083\n\n#136 0x41e988 in start /lib64/libc.so.2:2083\n\n#137 0x41e988 in start /lib64/libc.so.2:2083\n\n#138 0x41e988 in start /lib64/libc.so.2:2083\n\n#139 0x41e988 in start /lib64/libc.so.2:2083\n\n#140 0x41e988 in start /lib64/libc.so.2:2083\n\n#141 0x41e988 in start /lib64/libc.so.2:2083\n\n#142 0x41e988 in start /lib64/libc.so.2:2083\n\n#143 0x41e988 in start /lib64/libc.so.2:2083\n\n#144 0x41e988 in start /lib64/libc.so.2:2083\n\n#145 0x41e988 in start /lib64/libc.so.2:2083\n\n#146 0x41e988 in start /lib64/libc.so.2:2083\n\n#147 0x41e988 in start /lib64/libc.so.2:2083\n\n#148 0x41e988 in start /lib64/libc.so.2:2083\n\n#149 0x41e988 in start /lib64/libc.so.2:2083\n\n#150 0x41e988 in start /lib64/libc.so.2:2083\n\n#151 0x41e988 in start /lib64/libc.so.2:2083\n\n#152 0x41e988 in start /lib64/libc.so.2:2083\n\n#153 0x41e988 in start /lib64/libc.so.2:2083\n\n#154 0x41e988 in start /lib64/libc.so.2:2083\n\n#155 0x41e988 in start /lib64/libc.so.2:2083\n\n#156 0x41e988 in start /lib64/libc.so.2:2083\n\n#157 0x41e988 in start /lib64/libc.so.2:2083\n\n#158 0x41e988 in start /lib64/libc.so.2:2083\n\n#159 0x41e988 in start /lib64/libc.so.2:2083\n\n#160 0x41e988 in start /lib64/libc.so.2:2083\n\n#161 0x41e988 in start /lib64/libc.so.2:2083\n\n#162 0x41e988 in start /lib64/libc.so.2:2083\n\n#163 0x41e988 in start /lib64/libc.so.2:2083\n\n#164 0x41e988 in start /lib64/libc.so.2:2083\n\n#165 0x41e988 in start /lib64/libc.so.2:2083\n\n#166 0x41e988 in start /lib64/libc.so.2:2083\n\n#167 0x41e988 in start /lib64/libc.so.2:2083\n\n#168 0x41e988 in start /lib64/libc.so.2:2083\n\n#169 0x41e988 in start /lib64/libc.so.2:2083\n\n#170 0x41e988 in start /lib64/libc.so.2:2083\n\n#171 0x41e988 in start /lib64/libc.so.2:2083\n\n#172 0x41e988 in start /lib64/libc.so.2:2083\n\n#173 0x41e988 in start /lib64/libc.so.2:2083\n\n#174 0x41e988 in start /lib64/libc.so.2:2083\n\n#175 0x41e988 in start /lib64/libc.so.2:2083\n\n#176 0x41e988 in start /lib64/libc.so.2:2083\n\n#177 0x41e988 in start /lib64/libc.so.2:2083\n\n#178 0x41e988 in start /lib64/libc.so.2:2083\n\n#179 0x41e988 in start /lib64/libc.so.2:2083\n\n#180 0x41e988 in start /lib64/libc.so.2:2083\n\n#181 0x41e988 in start /lib64/libc.so.2:2083\n\n#182 0x41e988 in start /lib64/libc.so.2:2083\n\n#183 0x41e988 in start /lib64/libc.so.2:2083\n\n#184 0x41e988 in start /lib64/libc.so.2:2083\n\n#185 0x41e988 in start /lib64/libc.so.2:2083\n\n#186 0x41e988 in start /lib64/libc.so.2:2083\n\n#187 0x41e988 in start /lib64/libc.so.2:2083\n\n#188 0x41e988 in start /lib64/libc.so.2:2083\n\n#189 0x41e988 in start /lib64/libc.so.2:2083\n\n#190 0x41e988 in start /lib64/libc.so.2:2083\n\n#191 0x41e988 in start /lib64/libc.so.2:2083\n\n#192 0x41e988 in start /lib64/libc.so.2:2083\n\n#193 0x41e988 in start /lib64/libc.so.2:2083\n\n#194 0x41e988 in start /lib64/libc.so.2:2083\n\n#195 0x41e988 in start /lib64/libc.so.2:2083\n\n#196 0x41e988 in start /lib64/libc.so.2:2083\n\n#197 0x41e988 in start /lib64/libc.so.2:2083\n\n#198 0x41e988 in start /lib64/libc.so.2:2083\n\n#199 0x41e988 in start /lib64/libc.so.2:2083\n\n#200 0x41e988 in start /lib64/libc.so.2:2083\n\n#201 0x41e988 in start /lib64/libc.so.2:2083\n\n#202 0x41e988 in start /lib64/libc.so.2:2083\n\n#203 0x41e988 in start /lib64/libc.so.2:2083\n\n#204 0x41e988 in start /lib64/libc.so.2:2083\n\n#205 0x41e988 in start /lib64/libc.so.2:2083\n\n#206 0x41e988 in start /lib64/libc.so.2:2083\n\n#207 0x41e988 in start /lib64/libc.so.2:2083\n\n#208 0x41e988 in start /lib64/libc.so.2:2083\n\n#209 0x41e988 in start /lib64/libc.so.2:2083\n\n#210 0x41e988 in start /lib64/libc.so.2:2083\n\n#211 0x41e988 in start /lib64/libc.so.2:2083\n\n#212 0x41e988 in start /lib64/libc.so.2:2083\n\n#213 0x41e988 in start /lib64/libc.so.2:2083\n\n#214 0x41e988 in start /lib64/libc.so.2:2083\n\n#215 0x41e988 in start /lib64/libc.so.2:2083\n\n#216 0x41e988 in start /lib64/libc.so.2:2083\n\n#217 0x41e988 in start /lib64/libc.so.2:2083\n\n#218 0x41e988 in start /lib64/libc.so.2:2083\n\n#219 0x41e988 in start /lib64/libc.so.2:2083\n\n#220 0x41e988 in start /lib64/libc.so.2:2083\n\n#221 0x41e988 in start /lib64/libc.so.2:2083\n\n#222 0x41e988 in start /lib64/libc.so.2:2083\n\n#223 0x41e988 in start /lib64/libc.so.2:2083\n\n#224 0x41e988 in start /lib64/libc.so.2:2083\n\n#225 0x41e988 in start /lib64/libc.so.2:2083\n\n#226 0x41e988 in start /lib64/libc.so.2:2083\n\n#227 0x41e988 in start /lib64/libc.so.2:2083\n\n#228 0x41e988 in start /lib64/libc.so.2:2083\n\n#229 0x41e988 in start /lib64/libc.so.2:2083\n\n#230 0x41e988 in start /lib64/libc.so.2:2083\n\n#231 0x41e988 in start /lib64/libc.so.2:2083\n\n#232 0x41e988 in start /lib64/libc.so.2:2083\n\n#233 0x41e988 in start /lib64/libc.so.2:2083\n\n#234 0x41e988 in start /lib64/libc.so.2:2083\n\n#235 0x41e988 in start /lib64/libc.so.2:2083\n\n#236 0x41e988 in start /lib64/libc.so.2:2083\n\n#237 0x41e988 in start /lib64/libc.so.2:2083\n\n#238 0x41e988 in start /lib64/libc.so.2:2083\n\n#239 0x41e988 in start /lib64/libc.so.2:2083\n\n#240 0x41e988 in start /lib64/libc.so.2:2083\n\n#241 0x41e988 in start /lib64/libc.so.2:2083\n\n#242 0x41e988 in start /lib64/libc.so.2:2083\n\n#243 0x41e988 in start /lib64/libc.so.2:2083\n\n#244 0x41e988 in start /lib64/libc.so.2:2083\n\n#245 0x41e988 in start /lib64/libc.so.2:2083\n\n#246 0x41e988 in start /lib64/libc.so.2:2083\n\n#247 0x41e988 in start /lib64/libc.so.2:2083\n\n#248 0x41e988 in start /lib64/libc.so.2:2083\n\n#249 0x41e988 in start /lib64/libc.so.2:2083\n\n#250 0x41e988 in start /lib64/libc.so.2:2083\n\n#251 0x41e988 in start /lib64/libc.so.2:2083\n\n#252 0x41e988 in start /lib64/libc.so.2:2083\n\n#253 0x41e988 in start /lib64/libc.so.2:2083\n\n#254 0x41e988 in start /lib64/libc.so.2:2083\n\n#255 0x41e988 in start /lib64/libc.so.2:2083\n\n#256 0x41e988 in start /lib64/libc.so.2:2083\n\n#257 0x41e988 in start /lib64/libc.so.2:2083\n\n#258 0x41e988 in start /lib64/libc.so.2:2083\n\n#259 0x41e988 in start /lib64/libc.so.2:2083\n\n#260 0x41e988 in start /lib64/libc.so.2:2083\n\n#261 0x41e988 in start /lib64/libc.so.2:2083\n\n#262 0x41e988 in start /lib64/libc.so.2:2083\n\n#263 0x41e988 in start /lib64/libc.so.2:2083\n\n#264 0x41e988 in start /lib64/libc.so.2:2083\n\n#265 0x41e988 in start /lib64/libc.so.2:2083\n\n#266 0x41e988 in start /lib64/libc.so.2:2083\n\n#267 0x41e988 in start /lib64/libc.so.2:2083\n\n#268 0x41e988 in start /lib64/libc.so.2:2083\n\n#269 0x41e988 in start /lib64/libc.so.2:2083\n\n#270 0x41e988 in start /lib64/libc.so.2:2083\n\n#271 0x41e988 in start /lib64/libc.so.2:2083\n\n#272 0x41e988 in start /lib64/libc.so.2:2083\n\n#273 0x41e988 in start /lib64/libc.so.2:2083\n\n#274 0x41e988 in start /lib64/libc.so.2:2083\n\n#275 0x41e988 in start /lib64/libc.so.2:2083\n\n#276 0x41e988 in start /lib64/libc.so.2:2083\n\n#277 0x41e988 in start /lib64/libc.so.2:2083\n\n#278 0x41e988 in start /lib64/libc.so.2:2083\n\n#279 0x41e988 in start /lib64/libc.so.2:2083\n\n#280 0x41e988 in start /lib64/libc.so.2:2083\n\n#281 0x41e988 in start /lib64/libc.so.2:2083\n\n#282 0x41e988 in start /lib64/libc.so.2:2083\n\n#283 0x41e988 in start /lib64/libc.so.2:2083\n\n#284 0x41e988 in start /lib64/libc.so.2:2083\n\n#285 0x41e988 in start /lib64/libc.so.2:2083\n\n#286 0x41e988 in start /lib64/libc.so.2:2083\n\n#287 0x41e988 in start /lib64/libc.so.2:2083\n\n#288 0x41e988 in start /lib64/libc.so.2:2083\n\n#289 0x41e988 in start /lib64/libc.so.2:2083\n\n#290 0x41e988 in start /lib64/libc.so.2:2083\n\n#291 0x41e988 in start /lib64/libc.so.2:2083\n\n#292
```

Planner result: giveup

Thought

Observation  
{ 'success': true }

Executor#0 note

Thought

Observation  
You are the EXECUTOR for ARVO bug localization. You run heavy tools and MUST call finish\_task(summary=...) to return results.  
TOOLS  
- analyze\_crash\_log: -c<string> |refinement\_bundle=... |top\_n=<int>  
- extract\_json(response\_text, merge\_with\_last=<bool>)  
- view\_source(file\_path, start\_line=None, end\_line=None, max\_bytes=200000)  
- grep\_source(pattern, root='src', max\_results=200)  
- copy\_out(path, out\_path=None)  
- compare\_llm\_metrics(diff\_index\_json\_path, llm\_json\_path=None, chunks\_json\_path=None, line\_stack=0)  
- finish\_task(summary=<string>)  
RUNTIME OWNERSHIP  
- Do NOT run ASAN or GDB. The Planner owns runtime collection and has cached 'crash log'.  
- You MAY call llm\_analyze() with NO arguments to consume the cached crash\_log from the environment.  
REFINEMENT (augment-only, optional):  
- Use view\_source(grep) to fetch code slices for promising files/functions.  
- Then call llm\_analyze(refinement\_bundle={"source\_slices": [...], top\_n=<M>}) to GENERATE A NEW batch of candidates from these slices only.  
- This refinement MUST NOT narrow, edit, or reuse previous candidates; it produces up to exactly <M> fresh items.  
- After refinement, call extract\_json(response\_text, merge\_with\_last=false) so the new batch stays separate from the first batch.  
COMPARISON:  
- You may compare llm\_metrics multiple times. Evaluate the first JSON snapshot, optionally refine to produce a new batch; then evaluate again.  
- Provide the correct llm\_json\_path for each compare; do not call compare\_llm\_metrics without it.  
NO-OP/INVALID CALLS:  
- Do NOT re-run llm\_analyze repeatedly for the same inputs (at most one augment round).  
- Do NOT open files >200KB; keep total view\_source <=5 and grep\_source <=2.  
FINISH CONTRACT:  
- finish\_task(summary=...) must be a single-line, semicolon-separated key=val list including:  
ok=<true/false> |detection\_rate=<float> |localization\_rate=<float> |strict\_localization\_rate=<float> |  
improved=<true/false> |prev=<float> |curr=<float> |  
diff\_index\_json\_path=<paths> |chunks\_json\_path=<paths> |llm\_json\_path=<paths or none> |  
gdb=<unknown|bin=unknown>

Executor#0 note

Thought

Observation  
You have been delegated a task:  
Use the cached crash\_log produced by the Planner. Analyze, optionally augment with source views, evaluate twice, then return metrics.  
Steps:  
1) llm\_analyze() on args - consume cached crash\_log  
2) extract\_json(response\_text, merge\_with\_last=false) # first JSON snapshot (batch A)  
3) compare\_llm\_metrics(diff\_index\_json\_path=None, llm\_json\_path=<batch A JSON> | llm\_json\_path=<batch A JSON> | chunks\_json\_path=<batch A JSON> | line\_stack=0)  
4) SOURCE PEEK (optional, before second compare):  
- Parse crash\_log for file paths (e.g. -c<lines>) and function names  
- Include top\_suspects from batch A, focus on crash stack (top 3-5 frames)  
- For up to TOP\_K candidates (dedupe by file):  
- If file\_path known: view\_source(file\_path, start\_line=40, end\_line=40, max\_bytes=200000)  
- Else: grep\_source(pattern, root='src', max\_results=20) and view\_source(best\_match <=40  
Bedlines: <=5 view\_source and <=2 grep\_source; skip files >200KB  
5) REFINEMENT (augment-only, one pass)  
- llm\_analyze(refinement\_bundle={"source\_slices": [...], top\_n=<M>})  
- Generate a NEW batch (batch B) from slices only; does not modify batch A  
- extract\_json(response\_text, merge\_with\_last=false) # keep batch B separate  
6) compare\_llm\_metrics(diff\_index\_json\_path=None, llm\_json\_path=<batch B JSON> | llm\_json\_path=<batch B JSON> | steps=5 | reuse batch A's | llm\_json\_path=<batch A JSON> | chunks\_json\_path=<batch A JSON> | line\_stack=0)  
7) finish\_task(summary="ok<...> |detection\_rate=<...> |localization\_rate=<...> |  
strict\_localization\_rate=<...> |improved=<...> |prev=<...> |curr=<...> |  
diff\_index\_json\_path=<diff\_index.json> |chunks\_json\_path=<batch A JSON> |llm\_json\_path=<...> |  
gdb=<unknown|bin=unknown>")

Executor#0 call: llm\_analyze

Thought

{ llm\_analyze() }

Executor#0 result: llm\_analyze

Thought

Observation  
{ "ok": true, "mode": "crash", "text": "\n\n (in file\_path: '/src/libstdc++/src/iosfuzzer/sniffle\_fuzzer.cc')\n 'start\_line': 54, 'end\_line': 60, 'confidence': 'very likely', 'reason': 'vread uses memcpy with a negative size, indicating invalid parameters possibly due to bad input or internal state.'\n 'symbol': 'vread'\n } in (file\_path: '/src/libstdc++/src/common.c')\n 'start\_line': 857, 'end\_line': 869, 'confidence': 'likely', 'reason': 'header read might be calculating sizes based on invalid or corrupted header, leading to negative size in subsequent functions.'\n 'symbol': 'header\_read'\n } in (file\_path: '/src/libstdc++/src/cf/cf.c')\n 'start\_line': 395, 'end\_line': 404, 'confidence': 'likely', 'reason': 'caf\_read\_header initializes data from inputs, passing potentially invalid info leading to negative size errors down the call stack.'\n 'symbol': 'caf\_read\_header'\n } in (file\_path: '/src/libstdc++/src/sniffle.c')\n 'start\_line': 3180, 'end\_line': 3190, 'confidence': 'possible', 'reason': 'psf\_open\_file deals with file header initialization, might indirectly influence memory sizes that could be negative.'\n 'symbol': 'psf\_open\_file'\n } in (file\_path: '/src/libstdc++/src/common.c')\n 'start\_line': 988, 'end\_line': 998, 'confidence': 'possible', 'reason': 'psf\_binheader\_readf processes binary headers; corrupt input can lead to negative sizes in vread.'\n 'symbol': 'psf\_binheader\_readf'\n } in (file\_path: '/src/libstdc++/src/cf/cf.c')\n 'start\_line': 127, 'end\_line': 137, 'confidence': 'possible', 'reason': 'caf\_open potentially handles opening invalid files or inconsistent states leading to negative size calculations.'\n 'symbol': 'caf\_open'\n } in (file\_path: '/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp')\n 'start\_line': 569, 'end\_line': 599, 'confidence': 'unlikely', 'reason': 'Fuzzer's execution callback is related but indirect; actual fault likely lies in file handling logic instead.'\n 'symbol': 'ExecuteCallback'\n } in (file\_path: '/src/libstdc++/src/sniffle.c')\n 'start\_line': 440, 'end\_line': 450, 'confidence': 'unlikely', 'reason': 'psf\_open virtual may invoke incorrect handling of virtual files; less likely to be root cause of negative sizes.'\n 'symbol': 'sf\_open\_virtual'\n } in (file\_path: '/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp')\n 'start\_line': 20, 'end\_line': 30, 'confidence': 'very unlikely', 'reason': 'main function initializes framework; unlikely to cause negative size issues; more setup than core logic.'\n 'symbol': 'main'\n } in (file\_path: '/src/libstdc++/src/ios.c')\n 'start\_line': 311, 'end\_line': 321, 'confidence': 'unlikely', 'reason': 'psf\_read deals with reading data; highly indirect relation to the negative size issue during memcpy.'\n 'symbol': 'psf\_read'\n } in (file\_path: '/src/libstdc++/src/sniffle\_fuzzer.cc')\n 'start\_line': 54, 'end\_line': 60, 'confidence': 'very likely', 'reason': 'vread uses memcpy with a negative size, indicating invalid parameters possibly due to bad input or internal state.'\n 'symbol': 'vread'\n } in (file\_path: '/src/libstdc++/src/common.c')\n 'start\_line': 857, 'end\_line': 869, 'confidence': 'likely', 'reason': 'header read might be calculating sizes based on invalid or corrupted header, leading to negative size in subsequent functions.'\n 'symbol': 'header\_read'\n } in (file\_path: '/src/libstdc++/src/cf/cf.c')\n 'start\_line': 395, 'end\_line': 404, 'confidence': 'likely', 'reason': 'caf\_read\_header initializes data from inputs, passing potentially invalid info leading to negative size errors down the call stack.'\n 'symbol': 'caf\_read\_header'\n } in (file\_path: '/src/libstdc++/src/sniffle.c')\n 'start\_line': 3180, 'end\_line': 3190, 'confidence': 'possible', 'reason': 'psf\_open\_file deals with the header initialization, might indirectly influence memory sizes that could be negative.'\n 'symbol': 'psf\_open\_file'\n } in (file\_path: '/src/libstdc++/src/common.c')\n 'start\_line': 988, 'end\_line': 998, 'confidence': 'possible', 'reason': 'psf\_binheader\_readf processes binary headers; corrupt input can lead to negative sizes in vread.'\n 'symbol': 'psf\_binheader\_readf'\n } in (file\_path: '/src/libstdc++/src/cf/cf.c')\n 'start\_line': 127, 'end\_line': 137, 'confidence': 'possible', 'reason': 'caf\_open potentially handles opening invalid files or inconsistent states leading to negative size calculations.'\n 'symbol': 'caf\_open'\n } in (file\_path: '/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp')\n 'start\_line': 569, 'end\_line': 599, 'confidence': 'unlikely', 'reason': 'Fuzzer's execution callback is related but indirect; actual fault likely lies in the handling logic instead.'\n 'symbol': 'ExecuteCallback'\n } in (file\_path: '/src/libstdc++/src/sniffle.c')\n 'start\_line': 440, 'end\_line': 450, 'confidence': 'unlikely', 'reason': 'psf\_open virtual may invoke incorrect handling of virtual files; less likely to be root cause of negative sizes.'\n 'symbol': 'sf\_open\_virtual'\n } in (file\_path: '/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp')\n 'start\_line': 20, 'end\_line': 30, 'confidence': 'very unlikely', 'reason': 'main function initializes framework; unlikely to cause negative size issues; more setup than core logic.'\n 'symbol': 'main'\n } in (file\_path: '/src/libstdc++/src/ios.c')\n 'start\_line': 311, 'end\_line': 321, 'confidence': 'unlikely', 'reason': 'psf\_read deals with reading data; highly indirect relation to the negative size issue during memcpy.'\n 'symbol': 'psf\_read'\n } in (file\_path: '/home/bug\_localization\_agents/work\_arvo/25357/llm\_output/tmp.json')\n 'home/bug\_localization\_agents/work\_arvo/25357/llm\_output/tmp.json' }

Executor#0 call: compare\_llm\_metrics

Thought

{ compare\_llm\_metrics(llm\_json\_path="/home/bug\_localization\_agents/work\_arvo/25357/llm\_batch\_0001.json", chunks\_json\_path="/home/bug\_localization\_agents/work\_arvo/25357/case\_chunks.json", diff\_index\_json\_path="/home/bug\_localization\_agents/work\_arvo/25357/diff\_index.json", line\_stack=0) }

Executor#0 result: compare\_llm\_metrics

Thought

Executor#0 call: extract\_json

Thought

{ extract\_json(response\_text="\n\n (in file\_path: '/src/libstdc++/src/iosfuzzer/sniffle\_fuzzer.cc')\n 'start\_line': 54, 'end\_line': 60, 'confidence': 'very likely', 'reason': 'vread uses memcpy with a negative size, indicating invalid parameters possibly due to bad input or internal state.'\n 'symbol': 'vread'\n } in (file\_path: '/src/libstdc++/src/common.c')\n 'start\_line': 857, 'end\_line': 869, 'confidence': 'likely', 'reason': 'header read might be calculating sizes based on invalid or corrupted header, leading to negative size in subsequent functions.'\n 'symbol': 'header\_read'\n } in (file\_path: '/src/libstdc++/src/cf/cf.c')\n 'start\_line': 395, 'end\_line': 404, 'confidence': 'likely', 'reason': 'caf\_read\_header initializes data from inputs, passing potentially invalid info leading to negative size errors down the call stack.'\n 'symbol': 'caf\_read\_header'\n } in (file\_path: '/src/libstdc++/src/sniffle.c')\n 'start\_line': 3180, 'end\_line': 3190, 'confidence': 'possible', 'reason': 'psf\_open\_file deals with file header initialization, might indirectly influence memory sizes that could be negative.'\n 'symbol': 'psf\_open\_file'\n } in (file\_path: '/src/libstdc++/src/common.c')\n 'start\_line': 988, 'end\_line': 998, 'confidence': 'possible', 'reason': 'psf\_binheader\_readf processes binary headers; corrupt input can lead to negative sizes in vread.'\n 'symbol': 'psf\_binheader\_readf'\n } in (file\_path: '/src/libstdc++/src/cf/cf.c')\n 'start\_line': 127, 'end\_line': 137, 'confidence': 'possible', 'reason': 'caf\_open potentially handles opening invalid files or inconsistent states leading to negative size calculations.'\n 'symbol': 'caf\_open'\n } in (file\_path: '/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp')\n 'start\_line': 569, 'end\_line': 599, 'confidence': 'unlikely', 'reason': 'Fuzzer's execution callback is related but indirect; actual fault likely lies in file handling logic instead.'\n 'symbol': 'ExecuteCallback'\n } in (file\_path: '/src/libstdc++/src/sniffle.c')\n 'start\_line': 440, 'end\_line': 450, 'confidence': 'unlikely', 'reason': 'psf\_open virtual may invoke incorrect handling of virtual files; less likely to be root cause of negative sizes.'\n 'symbol': 'sf\_open\_virtual'\n } in (file\_path: '/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp')\n 'start\_line': 20, 'end\_line': 30, 'confidence': 'very unlikely', 'reason': 'main function initializes framework; unlikely to cause negative size issues; more setup than core logic.'\n 'symbol': 'main'\n } in (file\_path: '/src/libstdc++/src/ios.c')\n 'start\_line': 311, 'end\_line': 321, 'confidence': 'unlikely', 'reason': 'psf\_read deals with reading data; highly indirect relation to the negative size issue during memcpy.'\n 'symbol': 'psf\_read'\n } in (file\_path: '/home/bug\_localization\_agents/work\_arvo/25357/llm\_output/tmp.json')\n 'merge\_with\_last=false', 'symbol': 'psf\_read'\n } in (file\_path: '/home/bug\_localization\_agents/work\_arvo/25357/llm\_output/tmp.json')\n 'merge\_with\_last=false' }

Executor#0 result: extract\_json

Thought

Observation  
{ "ok": true, "mode": "llm\_predictions", "file\_path": "/src/libstdc++/src/iosfuzzer/sniffle\_fuzzer.cc", "start\_line": 54, "end\_line": 60, "confidence": "very likely", "reason": "vread uses memcpy with a negative size, indicating invalid parameters possibly due to bad input or internal state." "symbol": "vread" } in (file\_path: "/src/libstdc++/src/common.c", "start\_line": 857, "end\_line": 869, "confidence": "likely", "reason": "header read might be calculating sizes based on invalid or corrupted header, leading to negative size in subsequent functions." "symbol": "header\_read" } in (file\_path: "/src/libstdc++/src/cf/cf.c", "start\_line": 395, "end\_line": 404, "confidence": "likely", "reason": "caf\_read\_header initializes data from inputs, passing potentially invalid info leading to negative size errors down the call stack." "symbol": "caf\_read\_header" } in (file\_path: "/src/libstdc++/src/sniffle.c", "start\_line": 3180, "end\_line": 3190, "confidence": "possible", "reason": "psf\_open\_file deals with the header initialization, might indirectly influence memory sizes that could be negative." "symbol": "psf\_open\_file" } in (file\_path: "/src/libstdc++/src/common.c", "start\_line": 988, "end\_line": 998, "confidence": "possible", "reason": "psf\_binheader\_readf processes binary headers; corrupt input can lead to negative sizes in vread." "symbol": "psf\_binheader\_readf" } in (file\_path: "/src/libstdc++/src/cf/cf.c", "start\_line": 127, "end\_line": 137, "confidence": "possible", "reason": "caf\_open potentially handles opening invalid files or inconsistent states leading to negative size calculations." "symbol": "caf\_open" } in (file\_path: "/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp", "start\_line": 569, "end\_line": 599, "confidence": "unlikely", "reason": "Fuzzer's execution callback is related but indirect; actual fault likely lies in the handling logic instead." "symbol": "ExecuteCallback" } in (file\_path: "/src/libstdc++/src/sniffle.c", "start\_line": 440, "end\_line": 450, "confidence": "unlikely", "reason": "psf\_open virtual may invoke incorrect handling of virtual files; less likely to be root cause of negative sizes." "symbol": "sf\_open\_virtual" } in (file\_path: "/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp", "start\_line": 20, "end\_line": 30, "confidence": "very unlikely", "reason": "main function initializes framework; unlikely to cause negative size issues; more setup than core logic." "symbol": "main" } in (file\_path: "/src/libstdc++/src/ios.c", "start\_line": 311, "end\_line": 321, "confidence": "unlikely", "reason": "psf\_read deals with reading data; highly indirect relation to the negative size issue during memcpy." "symbol": "psf\_read" } in (file\_path: "/src/libstdc++/src/sniffle\_fuzzer.cc", "start\_line": 54, "end\_line": 60, "confidence": "very likely", "reason": "vread uses memcpy with a negative size, indicating invalid parameters possibly due to bad input or internal state." "symbol": "vread" } in (file\_path: "/src/libstdc++/src/common.c", "start\_line": 857, "end\_line": 869, "confidence": "likely", "reason": "header read might be calculating sizes based on invalid or corrupted header, leading to negative size in subsequent functions." "symbol": "header\_read" } in (file\_path: "/src/libstdc++/src/cf/cf.c", "start\_line": 395, "end\_line": 404, "confidence": "likely", "reason": "caf\_read\_header initializes data from inputs, passing potentially invalid info leading to negative size errors down the call stack." "symbol": "caf\_read\_header" } in (file\_path: "/src/libstdc++/src/sniffle.c", "start\_line": 3180, "end\_line": 3190, "confidence": "possible", "reason": "psf\_open\_file deals with the header initialization, might indirectly influence memory sizes that could be negative." "symbol": "psf\_open\_file" } in (file\_path: "/src/libstdc++/src/common.c", "start\_line": 988, "end\_line": 998, "confidence": "possible", "reason": "psf\_binheader\_readf processes binary headers; corrupt input can lead to negative sizes in vread." "symbol": "psf\_binheader\_readf" } in (file\_path: "/src/libstdc++/src/cf/cf.c", "start\_line": 127, "end\_line": 137, "confidence": "possible", "reason": "caf\_open potentially handles opening invalid files or inconsistent states leading to negative size calculations." "symbol": "caf\_open" } in (file\_path: "/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp", "start\_line": 569, "end\_line": 599, "confidence": "unlikely", "reason": "Fuzzer's execution callback is related but indirect; actual fault likely lies in the handling logic instead." "symbol": "ExecuteCallback" } in (file\_path: "/src/libstdc++/src/sniffle.c", "start\_line": 440, "end\_line": 450, "confidence": "unlikely", "reason": "psf\_open virtual may invoke incorrect handling of virtual files; less likely to be root cause of negative sizes." "symbol": "sf\_open\_virtual" } in (file\_path: "/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp", "start\_line": 20, "end\_line": 30, "confidence": "very unlikely", "reason": "main function initializes framework; unlikely to cause negative size issues; more setup than core logic." "symbol": "main" } in (file\_path: "/src/libstdc++/src/ios.c", "start\_line": 311, "end\_line": 321, "confidence": "unlikely", "reason": "psf\_read deals with reading data; highly indirect relation to the negative size issue during memcpy." "symbol": "psf\_read" } in (file\_path: "/home/bug\_localization\_agents/work\_arvo/25357/llm\_output/tmp.json") }

Executor#0 call: compare\_llm\_metrics

Thought

{ compare\_llm\_metrics(llm\_json\_path="/home/bug\_localization\_agents/work\_arvo/25357/llm\_batch\_0001.json", chunks\_json\_path="/home/bug\_localization\_agents/work\_arvo/25357/case\_chunks.json", diff\_index\_json\_path="/home/bug\_localization\_agents/work\_arvo/25357/diff\_index.json", line\_stack=0) }

Executor#0 result: compare\_llm\_metrics

Thought







Executor#0 note Thought

Observation

You are the EXECUTOR for ARVO bug localization. You run heavy tools and MUST call finish\_task(summary=...) to return results. TOOLS: - llm\_analyze (crash\_log=<string> [refinement\_bundle=..., top\_n=<int>)

Executor#0 note Thought

Observation

You have been delegated a task. Use the cached crash\_log produced by the Planner. Analyze, optionally refine with source views, then evaluate and return metrics. Steps: 1) llm\_analyze() # no args -> consume cached crash\_log

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: extract\_json Thought

Executor#0 result: extract\_json Thought

Executor#0 call: view\_source Thought

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: extract\_json Thought

Executor#0 result: extract\_json Thought

Executor#0 call: extract\_json Thought

Executor#0 result: extract\_json Thought

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: extract\_json Thought

Executor#0 result: extract\_json Thought

Executor#0 call: extract\_json Thought

Executor#0 result: extract\_json Thought

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: llm\_analyze Thought

Executor#0 result: llm\_analyze Thought

Executor#0 call: extract\_json Thought

Executor#0 result: extract\_json Thought

Executor#0 call: extract\_json Thought

Executor#0 result: extract\_json Thought

Observation

Executor#0 call: view\_source Thought

Executor#0 result: view\_source Thought

