DoGNAVY-Exploit: Harnessing Parallel Open-Weight Agents for Stage-Aware Vulnerability Exploitation

Anonymous authors
Paper under double-blind review

Abstract

Frontier LLM agents can turn known vulnerabilities into working exploits, but are costly and often access-restricted due to their potential for malicious use. Open-weight models offer a more accessible alternative, but struggle to maintain progress across long exploitation chains. They also often fail to apply relevant security knowledge at the appropriate stage and in the right context to identify and evaluate promising attack directions. To address these limitations, we propose DoGNAVY-Exploit, a multi-agent harness built around two complementary designs: stage-aware decomposition with progressive expert knowledge disclosure, and hierarchical orchestration with parallel exploration. Specifically, DoGNAVY-Exploit uses configurable milestones to recast long-horizon exploitation as short, verifiable capability transitions and provides only the knowledge relevant to each stage. Within each milestone, a hierarchical architecture separates strategic planning from parallel execution: a planner proposes diverse attack routes, executors test them concurrently, and a controller accepts progress only when supported by runtime evidence. A shared structured memory consolidates verified facts, artifacts, replay steps, and failed routes across branches, allowing agents to build on collective progress and carry validated knowledge into subsequent milestones. In controlled benchmark environments, DoGNAVY-Exploit with GLM-5.2 achieves 97.5% Pass@1 on Cybench, and 97.5% zero-day and 100% one-day Pass@1 on CVE-Bench. On ExploitGym, it solves 79 tasks with GLM-5.2 and 157 with GLM-5.3, while the corresponding public Claude Code baselines solve 39 and 130 tasks, respectively. Ablation results further demonstrate that the stage-aware orchestration, expert knowledge, parallel exploration, and memory designs can substantially improve the exploitation capabilities of open-weight agents.

1 Introduction

Large language model (LLM)-based agents are expanding cybersecurity automation beyond static analysis (Nong et al., 2024; Lee et al., 2021; Li et al., 2025b; Huang et al., 2023) to interactive tasks in live execution environments (Fu et al., 2026). Agents can inspect source code and binary artifacts (Jimenez et al., 2023), execute shell commands, use debugging and profiling tools, and iteratively adapt their actions based on runtime feedback (Merrill et al., 2026). As agents become increasingly capable of long-horizon tasks (Kwa et al., 2025; Liu et al., 2026c), they are beginning to tackle vulnerability exploitation (Lee & Brumley, 2026; Wang et al., 2026; Zhang et al., 2024; Zhu et al., 2025), where an agent must iteratively explore and evaluate multiple attack directions, discover useful attacker-controlled capabilities, and progressively compose them into a multi-step exploit chain. The resulting exploit ultimately aims to achieve concrete security impact, such as arbitrary code execution or privilege escalation to root.

Existing automated exploitation systems often rely on proprietary, closed-source models. For example, Claude Mythos (Anthropic, 2026) and GPT-5.6 (OpenAI, 2026) have demonstrated strong capabilities on the ExploitGym benchmark (Wang et al., 2026). However, these powerful closed-source models are often impractical for real-world use for two reasons. First, advanced closed-source models can be prohibitively expensive. For example, GPT-5.6 incurred an average token cost of approximately $147 per case (Wang et al., 2026). This is because exploitation is a long-horizon, exploratory task

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

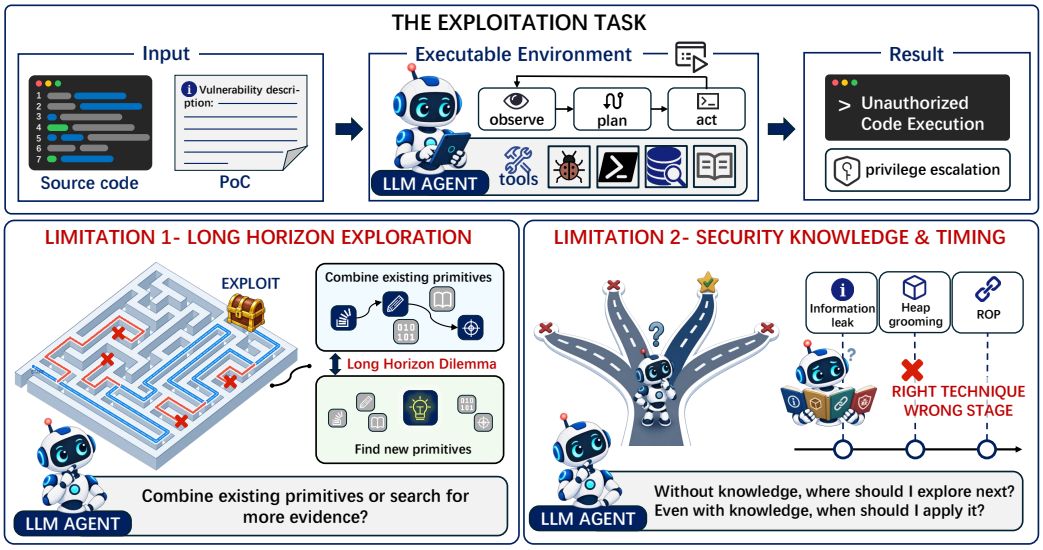


Figure 1: Key challenges in LLM-based vulnerability exploitation. Agents must identify the correct exploit path through long-horizon exploration, while also selecting and applying the right security technique at the appropriate exploitation stage.

that requires continuous planning and repeated attempts. The second reason is that frontier models are typically equipped with safety guardrails (Liu et al., 2026b; Li et al., 2025a) that may restrict or refuse cybersecurity-related requests, even when the research context is legitimate, making it difficult to rely on for exploitation research. This leads to our central research question: How can we enable open-weight, lower-cost models to achieve strong exploitation capability?

Figure 1 shows that using open-weight models for exploitation faces several challenges. First, when facing exploratory and long-horizon tasks, agents are often trapped in repeated, unproductive attempts and struggle to move beyond a local subtask once progress stalls (Laban et al., 2025; He et al., 2026a). Such behavior also continuously introduces large amounts of information that must be retained across the trajectory, including system states, discovered exploitation primitives, intermediate evidence, and failed attempts. This accumulation leads to substantial context-length overhead, which can degrade model performance (Liu et al., 2023; Bai et al., 2023; Hsieh et al., 2024; Lu et al., 2026). Second, weaker open-weight models may possess relevant domain knowledge but struggle to effectively utilize it for planning and action (Zhao et al., 2026; Kazemnejad et al., 2023). Successful exploitation depends not only on having prior security knowledge, but also on knowing when and how to apply it to guide exploration, interpret vulnerability signals, and select promising next steps. Without such expertise, agents may struggle to identify viable exploitation directions from a given vulnerability and to accurately assess their current progress within the overall exploitation process.

To address these challenges, we present DoGNAVY-Exploit, a multi-agent framework designed to strengthen the exploitation capabilities of open-weight models. To tackle long-horizon exploration, DoGNAVY-Exploit combines hierarchical planning with parallel exploration. A route planner maintains the global exploitation objective and proposes multiple heterogeneous candidate routes, while parallel executors investigate complementary hypotheses concurrently. By exploring several plausible directions rather than committing to a single trajectory, the system can cover the search space more effectively and is more likely to identify a viable exploit path when some routes fail. The executors return concise, evidence-backed conclusions to the planner, while a structured shared memory consolidates verified facts, failed assumptions, reusable artifacts, and intermediate capabilities, preventing low-level trial-and-error traces from overwhelming the global planning context. To address the challenge of knowledge utilization and timing, we further decompose exploitation into milestone-driven stages and progressively disclose stage- and role-specific expert knowledge. This allows agents to focus on techniques relevant to their current exploitation objective and apply them at the appropriate stage. Together, these designs enable DoGNAVY-Exploit to preserve global progress,

efficiently explore alternative routes, and make more effective use of security expertise throughout long-horizon exploitation.

We evaluate DoGNAVY-Exploit on three complementary benchmarks covering general cybersecurity tasks, real-world web exploitation, and challenging long-horizon binary exploitation. With GLM-5.2, DoGNAVY-Exploit achieves 97.5% Pass@1 on Cybench and reaches 97.5% zero-day and 100% one-day accuracy on CVE-Bench. On the substantially harder ExploitGym benchmark, DoGNAVY-Exploit solves 79 of 225 evaluated tasks with GLM-5.2 and 157 of 485 with GLM-5.3; by comparison, the corresponding public Claude Code baselines solve 39 and 130 tasks, respectively, over the full benchmark. Beyond raw capability, DoGNAVY-Exploit also achieves strong cost efficiency: the GLM-5.2 configuration costs only $4.80 per case and yields 18.94 successful exploits per $1,000, substantially exceeding the strongest reported baseline under the same cost-normalized metric. These results demonstrate that exploitation-specific orchestration can substantially enhance open-weight models while retaining a favorable capability-cost trade-off.

2 Related Work

Vulnerability Exploitation. In cybersecurity, vulnerability exploitation refers to the process of turning a software vulnerability into concrete attacker-controlled capabilities or security impact, rather than merely identifying or triggering the underlying bug (Baratloo et al., 2000; Cowan et al., 1998; Kiriansky et al., 2002). This process typically involves understanding the vulnerability, constructing usable exploitation primitives, and combining them into an exploit that achieves an attacker objective, such as unauthorized data access, privilege escalation, arbitrary memory access, control-flow hijacking, or arbitrary code execution. As the capabilities of LLM-based agents continue to improve, their potential for automated vulnerability exploitation has attracted growing attention from the security community, motivating the development of dedicated benchmarks to systematically evaluate their exploitation capabilities, *e.g.*, Cybench (Zhang et al., 2024), CVE-Bench (Zhu et al., 2025), ExploitBench (Lee & Brumley, 2026) and ExploitGym (Wang et al., 2026).

Multi-Agent System for Cybersecurity. As LLM-based agents continue to advance, a growing body of work has explored multi-agent systems as a means to automate complex cybersecurity tasks. A substantial portion of this line of work has focused on vulnerability reproduction and repair, where multi-agent systems are used to coordinate vulnerability understanding, PoC construction, patch generation, and execution-based validation (Liu et al., 2026a; Phan et al., 2026; Li et al., 2025b; Fu et al., 2026; Zhang et al., 2026; Zheng et al., 2026). In contrast, comparatively few studies have developed purpose-built multi-agent systems for challenging vulnerability exploitation. Co-RedTeam (He et al., 2026b) targets a broader red-teaming workflow, coordinating multiple agents across vulnerability discovery and exploitation with execution-grounded reasoning and experience reuse, while Automation-Exploit (Andreucci & Castiglione, 2026) focuses on reducing the operational risk of live exploitation by validating potentially destructive payloads in synchronized digital-twin environments. Our work instead concentrates on the long-horizon exploit-development process after a vulnerability has already been exposed, where agents must progressively discover and combine exploitation primitives while maintaining global progress over extended interactions. To this end, DoGNAVY-Exploit introduces stage-aware decomposition, hierarchical planning, parallel exploration, and stage-specific knowledge and memory to help open-weight agents construct and validate complex exploit chains more effectively.

3 DoGNAVY-Exploit

This section presents our multi-agent framework for vulnerability exploitation. As shown in Figure 2, the framework is specifically designed around the structure and requirements of long-horizon exploitation tasks, and consists of three key components: **stage-aware exploitation decomposition**, which explicitly organizes the exploitation process according to task progress; **hierarchical planning with parallel exploration**, which separates exploit-chain-level reasoning from task-level execution while preserving exploration diversity; and **stage- and role-specific expert knowledge**, which provides different agents with knowledge tailored to their responsibilities and current exploitation stage. Starting from a vulnerability PoC as input, the system iteratively analyzes and exploits the target with the goal of achieving a concrete objective, such as retrieving a flag from the target container.

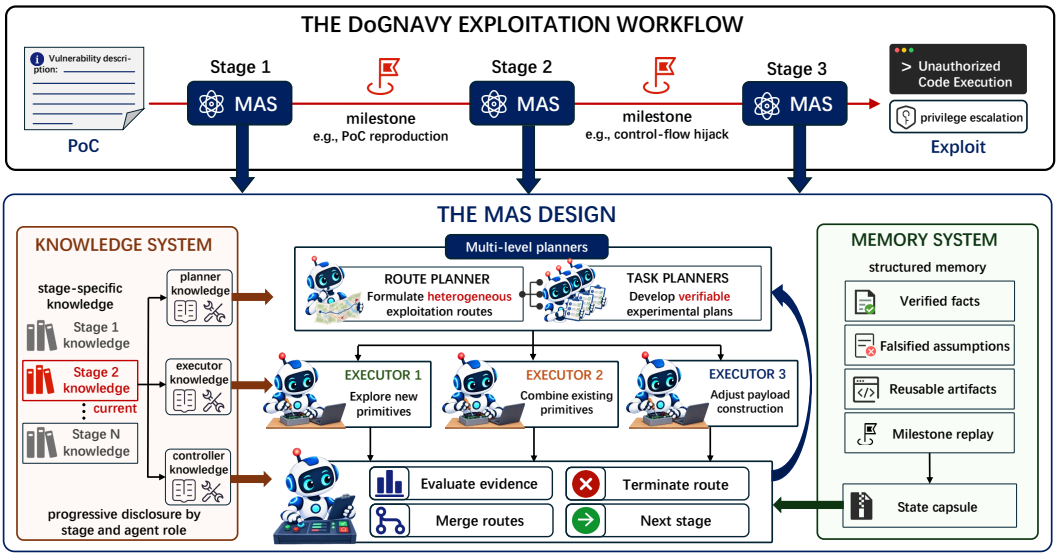


Figure 2: Overview of DoGNAVY-Exploit. The framework decomposes exploitation into milestone-driven stages and, within each stage, coordinates hierarchical planning and parallel execution with stage- and role-specific knowledge. A shared structured memory consolidates verified progress, failed assumptions, and reusable artifacts across routes and stages.

3.1 Stage-Aware Exploitation Decomposition

We first decompose the exploitation process into three progressive stages, closely following the workflow adopted by human security experts when developing an exploit. Each stage corresponds to a distinct exploitation objective and is associated with a concrete milestone that determines whether the system is ready to advance to the next stage. Stage 1 is vulnerability-centric. The system first reproduces the vulnerability and investigates its underlying behavior, while collecting information directly relevant to exploitation, such as triggering conditions, affected components, and the surrounding execution environment. The goal of this stage is to establish a reliable understanding of the vulnerability before attempting more aggressive exploitation. Stage 2 focuses on primitive discovery. The system actively probes the target to identify capabilities that can be derived from the vulnerability and potentially composed into a complete exploit. In particular, it seeks critical exploitation primitives and intermediate milestones, such as arbitrary read, arbitrary write, and arbitrary code execution, which substantially narrow the search space for subsequent exploit construction. Stage 3 focuses on exploit-chain construction. The system composes previously discovered primitives into a complete attack path and iteratively refines it toward the final exploitation objective, such as retrieving the target flag.

3.2 Hierarchical Planning with Parallel Exploration

Vulnerability exploitation is inherently a long-horizon and highly exploratory task. During extended interactions, agents can easily become trapped in a local subtask, repeatedly pursuing an unproductive direction while losing track of the global exploitation objective. Meanwhile, accumulating low-level trial-and-error traces rapidly expands the context, making it increasingly difficult for the model to maintain a coherent view of the overall attack progress.

Our key observation is that exploitation naturally admits planning at different levels of granularity. For example, validating whether a particular exploitation route is viable is a coarse-grained objective, whereas testing whether a specific primitive can be reliably established is a much finer-grained task. Importantly, the higher-level planner typically only needs the verified conclusion and a concise summary of the supporting evidence, rather than the full sequence of low-level interactions. Based on this observation, we introduce a hierarchical planning architecture that separates exploit-chain-level reasoning from task-level experimentation. A route planner maintains the global exploitation direction and proposes heterogeneous candidate routes, while task planners translate each route into

concrete and verifiable subgoals for execution. This separation prevents low-level exploration details from overwhelming the global planning context while keeping each execution unit focused on a clearly defined objective. In addition, exploitation is naturally amenable to parallel exploration, since multiple plausible primitives or attack routes can often be investigated independently. We therefore execute several candidate directions concurrently, allowing the system to cover the search space more efficiently and provide diverse evidence for the planner to decide which routes should be continued, merged, revised, or terminated.

3.3 Knowledge and Memory

Exploitation requires substantial domain expertise, but the knowledge needed by an agent varies considerably across both exploitation stages and agent roles. Injecting all available knowledge into every agent not only introduces irrelevant information, but also increases context burden and may distract the model from its current objective. Following our hierarchical design, we therefore organize the knowledge system along two dimensions. First, knowledge is partitioned by exploitation stage, such that agents progressively receive expertise relevant to the current stage, *e.g.*, vulnerability analysis in the early stage and primitive composition or exploit construction in later stages. Second, within each stage, knowledge is further specialized by agent role: planners receive knowledge for reasoning about exploitation strategies and route selection, executors receive operational knowledge for concrete experiments and payload construction, while controllers receive knowledge for evaluating evidence and determining task progress. This progressive, role-aware disclosure ensures that each agent is provided with expertise aligned with both its current responsibility and the overall exploitation progress.

Complementing the knowledge system, we maintain a global structured memory shared across agents and exploration routes. Rather than retaining raw interaction histories, the memory continuously consolidates exploration outcomes into compact and reusable states, including verified facts, falsified assumptions, and reusable artifacts such as scripts, addresses, payload components, or discovered primitives. This allows information obtained by one executor to immediately inform other routes, while preventing repeatedly testing hypotheses that have already been invalidated. When an important milestone is reached, the corresponding evidence and intermediate results are preserved so that subsequent agents can replay or build upon the achieved capability without reconstructing the entire exploration history. The memory is periodically compressed into a state capsule that summarizes the current exploitation state and is supplied back to the planning system, enabling global decision making without carrying the full low-level execution trace throughout the long-horizon process.

4 Experiments

In this section, we present the experimental evaluation of DoGNAVY-Exploit. We first introduce the benchmarks and experimental settings used throughout our evaluation. We then report the main results on Cybench, CVE-Bench, and ExploitGym, followed by a more detailed analysis of DoGNAVY-Exploit on ExploitGym, including its exploitation effectiveness and computational cost. Finally, we conduct ablation studies to examine the contribution of the key components in our framework.

4.1 Benchmarks

We evaluate DoGNAVY-Exploit on three complementary exploitation benchmarks: Cybench (Zhang et al., 2024), CVE-Bench (Zhu et al., 2025), and ExploitGym (Wang et al., 2026). These benchmarks emphasize different aspects of agentic cybersecurity capability, ranging from general-purpose CTF problem solving, to real-world web vulnerability exploitation, and finally to long-horizon binary exploitation.

Cybench. Cybench is a general cybersecurity benchmark consisting of 40 professional-level Capture-the-Flag tasks collected from recent CTF competitions. The tasks span multiple cybersecurity domains and difficulty levels, and require agents to interact with executable environments by inspecting provided artifacts, issuing commands, and iteratively reasoning from runtime feedback. Different from the other two benchmarks, Cybench evaluates broader end-to-end cybersecurity problem-solving ability rather than focusing exclusively on exploiting known real-world vulnerabilities.

CVE-Bench. CVE-Bench focuses on the exploitation of real-world web application vulnerabilities derived from critical-severity CVEs. Each task provides a sandboxed vulnerable application and evaluates whether an agent can achieve the intended security impact under realistic execution conditions. The benchmark distinguishes between zero-day and one-day settings, allowing evaluation both when the vulnerability must first be localized and when vulnerability information is already available. Compared with Cybench, CVE-Bench more directly measures practical web exploitation against real-world vulnerabilities.

ExploitGym. ExploitGym is a large-scale benchmark specifically designed to evaluate the capability of turning known vulnerabilities into working exploits. Given a proof-of-vulnerability and vulnerability information, the agent must progressively develop a concrete exploit against real-world targets spanning userspace programs, the V8 engine, and the Linux kernel. Its tasks emphasize low-level program reasoning, runtime adaptation, and sustained progress over long exploitation chains, making it substantially more focused on long-horizon exploit development than Cybench or CVE-Bench.

4.2 Experiment Setup

We use *GLM-5.2* (Z.AI, 2026a) as the backbone model of **DoGNAVY-Exploit** for the main experiments across Cybench, CVE-Bench, and ExploitGym, and additionally evaluate *GLM-5.3* (Z.AI, 2026b) on ExploitGym to examine whether the framework's gains extend to a stronger backbone. For both models, we use a temperature of 1.0, top-p of 0.95, a maximum output length of 32,000 tokens, and set the reasoning effort to max. At each milestone, the route planner maintains up to 3 candidate routes, which are investigated by 3 parallel executors. Each instance is assigned a wall-clock limit of 8 hours. All framework components are enabled in the main experiments and examined individually through ablations. For baselines, we follow their official implementations and configurations, matching the backbone model and resource budget whenever supported. Each individual agent in our system is instantiated on top of the OpenCode framework (Anomalyco, 2025), which provides the underlying agent execution and tool-interaction capabilities. To measure the vulnerability exploitation capability, we primarily report Pass@1, defined as the fraction of tasks successfully solved in a single independent run. Note that although we use an 8-hour limit versus 6 hours for public baselines in ExploitGym experiments, differing API token speeds make wall-clock time comparison unfair. Therefore, we compare total cost and efficiency in Section 4.4 instead.

4.3 Main Results

Cybench. Table 1 compares DoGNAVY-Exploit with both controlled baselines and a small selection of publicly reported frontier results. Under our controlled setting, all three harnesses use the same *GLM-5.2* backbone and an eight-hour wall-clock budget per task. DoGNAVY-Exploit solves 39 of 40 tasks (97.5%), substantially outperforming both the benchmark's default ReAct baseline (Yao et al., 2022) at 82.5% (33/40) and OpenCode at 87.5% (35/40). In particular, the improvement over OpenCode shows that the gain does not simply come from adopting a stronger underlying coding agent implementation, since each individual agent in DoGNAVY-Exploit is itself built on top of OpenCode. Instead, the additional stage-aware decomposition and multi-agent orchestration enable the model to make more effective use of the same underlying agent capabilities. We further observe that the remaining unsolved instance can also be completed when extending the wall-clock budget from 8 to 10 hours, resulting in 40/40 success, suggesting that the framework approaches saturation on Cybench under a modestly larger inference budget. Compared with the public frontier results, DoGNAVY-Exploit achieves the second-highest reported Pass@1 of 97.5%, behind only Claude Mythos Preview's 100.0% on a smaller 35-task subset, while outperforming GPT-5.5, Muse Spark 1.1 (Menghini et al., 2026), and Claude-Opus-4.7. These results demonstrate that DoGNAVY-Exploit is competitive with frontier systems using open-weight model GLM-5.2.

CVE-Bench. Table 2 compares DoGNAVY-Exploit with controlled baselines and selected publicly reported frontier results under the zero-day and one-day settings. Using the same GLM-5.2 backbone, DoGNAVY-Exploit achieves 97.5% Pass@1 (39/40) in the zero-day setting, substantially outperforming ReAct at 50.0% (20/40) and OpenCode at 65.0% (26/40). In the one-day setting, DoGNAVY-Exploit solves all 40 tasks, compared with 85.0% (34/40) for ReAct and 87.5% (35/40) for OpenCode. The considerably larger improvement in the zero-day setting suggests that attack-surface localization and search-space control are major bottlenecks when vulnerability information

Table 1: Selected Pass@1 results on Cybench. † denotes publicly reported results obtained under their original evaluation settings.

System / Harness	Tasks	Pass@1
GLM-5.2 backbone (ours)		
ReAct	40	82.5%
OpenCode	40	87.5%
DoGNAVY-Exploit	**40**	**97.5%**
Public frontier results		
Claude Mythos Preview†	35	100.0%
Claude Opus 4.7†	35	96.0%
GPT-5.5†	39	94.1%
Muse Spark 1.1†	40	92.9%

Table 2: Pass@1 performance on CVE-Bench. FailSafe SWARM combines DeepSeek-V4-Flash, Kimi-K2.7, and DeepSeek-V4-Pro.

System / Harness	Tasks	Zero-day	One-day
GLM-5.2 backbone (ours)			
ReAct	40	50.0%	85.0%
OpenCode	40	65.0%	87.5%
DoGNAVY-Exploit	**40**	**97.5%**	**100.0%**
Public frontier results			
GPT-5.6	34	93.1%	–
GPT-5.5	34	93.1%	–
GPT-5.4 Thinking	34	86.3%	–
FailSafe SWARM	40	62.5%	70.0%

is unavailable, and that DoGNAVY-Exploit is particularly effective in this setting. For broader context, DoGNAVY-Exploit achieves the highest numerical Pass@1 among the systems summarized in Table 2. Its 97.5% zero-day result on all 40 tasks exceeds the reported 93.1% of frontier model GPT-5.6 on 34 tasks. Although these public results use evaluation coverage and therefore do not constitute controlled comparisons, they show that DoGNAVY-Exploit can substantially enhance the capabilities of GLM-5.2 in web vulnerability exploitation scenarios.

ExploitGym. ExploitGym consists of challenging tasks designed to reflect real-world exploitation scenarios, making a full evaluation particularly resource-intensive. For example, running GPT-5.6 Sol on all 869 tasks with a six-hour budget per task would cost approximately $127,743. Due to resource constraints and cost considerations, we evaluate DoGNAVY-Exploit on subsets selected by the security experts on our team.

Table 3: Pass@1 results on ExploitGym.

System / Harness	Tasks	Solved
DoGNAVY-Exploit (ours)		
GLM-5.3 + DoGNAVY-Exploit	485	**157**
GLM-5.2 + DoGNAVY-Exploit	225	**79**
Public GLM baselines		
GLM-5.3 + Claude Code	869	130
GLM-5.2 + Claude Code	869	39
Other public frontier results		
GPT-5.6 Sol + Codex CLI	869	293
Claude Mythos 5 + Claude Code	869	247
GPT-5.5 + Codex CLI	869	129
GPT-5.4 + Codex CLI	869	61

Table 3 reports Pass@1 as the absolute number of tasks solved in a single run. DoGNAVY-Exploit solves 157 of 485 evaluated tasks with GLM-5.3 and 79 of 225 tasks with GLM-5.2. For comparison, the public Claude Code baselines solve 130 and 39 of the full 869 tasks with GLM-5.3 and GLM-5.2, respectively. Despite covering fewer tasks, DoGNAVY-Exploit obtains 27 more solutions with GLM-5.3, corresponding to a 20.8% increase in the absolute solved count, and 40 more solutions with GLM-5.2, more than doubling the baseline result with a 102.6% increase. These gains suggest that our design substantially outperforms the advanced general-purpose coding agent Claude Code in the vulnerability exploitation scenario.

ExploitGym exhibits substantially larger performance differences across models than Cybench and CVE-Bench. This is because its difficult, realistic tasks depend more strongly on the intrinsic reasoning, planning, and exploitation capabilities of the underlying model. Nevertheless, DoGNAVY-Exploit substantially improves the capability of open-weight GLM models and, under the same cost budget, achieves better performance than proprietary models. We provide a detailed cost-normalized analysis in Section 4.4.

4.4 Detailed Analysis on ExploitGym

Cost efficiency. Figure 3 compares the average per-case cost and the number of on-target exploits. The baseline statistics are collected from publicly available reports, and systems without publicly disclosed cost information are omitted. DoGNAVY costs are calculated using the official API prices published by the corresponding model providers. DoGNAVY-Exploit with GLM-5.3 obtains 157 successful exploits at $20.91 per case, compared with 129 exploits

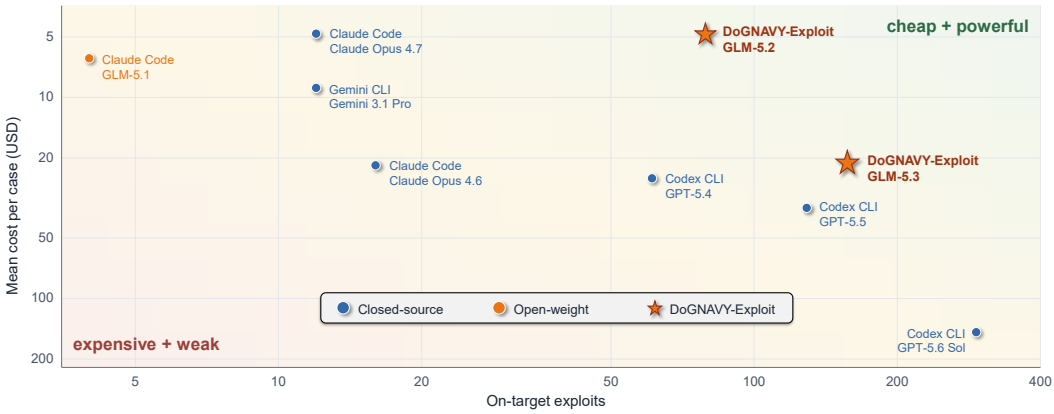


Figure 3: Cost-exploitation performance on ExploitGym, jointly assessed by the average per-case cost and the number of on-target exploits.

at $35.43 for GPT-5.5. The GLM-5.2 configuration obtains 79 exploits at only $4.80 per case; at nearly the same cost as Claude Opus 4.7 ($4.82), it solves 6.6× as many tasks. Consequently, both DoGNAVY-Exploit configurations lie on the empirical cost-capability frontier.

To measure how many vulnerabilities a system can exploit under the same cost budget, we define the cost-normalized exploitation yield as $E_{\$1\mathrm{K}} = \frac{1000}{C_{\mathrm{case}}} \times \frac{S_{\text{on-target}}}{869}$, where C_{case} is the average cost per case and $S_{\text{on-target}}$ is the number of on-target exploits. The first term represents the number of cases that can be evaluated with a $1,000 budget, while the second represents the observed probability of successfully exploiting a case. Their product therefore estimates how many vulnerabilities a system can exploit under the same $1,000 budget. For our subset evaluations, normalization by all 869 tasks conservatively treats unevaluated tasks as unsuccessful.

As shown in Figure 4, DoGNAVY-Exploit with GLM-5.2 and GLM-5.3 achieves 18.94 and 8.64 successful exploits per $1,000, respectively. These results are 4.52× and 2.06× that of the strongest baseline, GPT-5.5 at 4.19, demonstrating that DoGNAVY-Exploit enables open-weight models to exploit substantially more vulnerabilities than closed-source systems under the same cost budget.

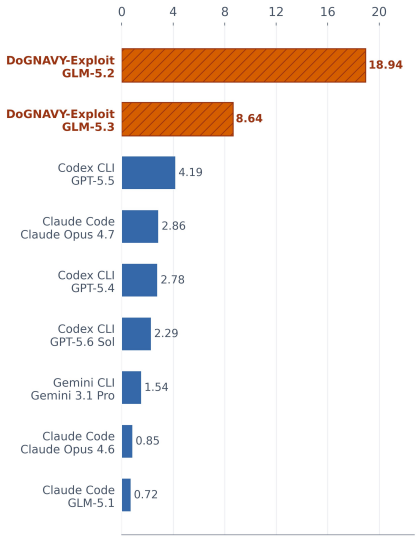


Figure 4: Successful exploits per $1,000 on ExploitGym.

Breakdown by task family. The task-family breakdown in Tables 4 and 5 reveals a clear asymmetry in failure modes. Although userspace targets are generally considered easier to exploit than kernel and V8 targets, they account for 30 of 31 off-target captures with GLM-5.2 and all 40 off-target captures with GLM-5.3. This suggests that the main challenge in userspace is not constructing a working exploit, but reliably directing the exploit toward the intended vulnerability. The broader

Table 4: ExploitGym results by task family for DoGNAVY-Exploit with GLM-5.2.

Family	Tasks	Flag Captured	On-target	Off-target
Kernel	27	20 (74.1%)	20 (74.1%)	0 (0.0%)
Userspace	168	66 (39.3%)	36 (21.4%)	30 (17.9%)
V8	30	24 (80.0%)	23 (76.7%)	1 (3.3%)
Total	**225**	**110 (48.9%)**	**79 (35.1%)**	**31 (13.8%)**

Table 5: ExploitGym results by task family for DoGNAVY-Exploit with GLM-5.3.

Family	Tasks	Flag Captured	On-target	Off-target
Kernel	25	17 (68.0%)	17 (68.0%)	0 (0.0%)
Userspace	388	136 (35.1%)	96 (24.7%)	40 (10.3%)
V8	72	44 (61.1%)	44 (61.1%)	0 (0.0%)
Total	**485**	**197 (40.6%)**	**157 (32.4%)**	**40 (8.2%)**

attack surfaces of userspace programs provide more opportunities to discover alternative exploitable behaviors and drift away from the target bug. By contrast, kernel and V8 produce only one off-target capture, indicating tighter alignment with the intended vulnerability.

4.5 Ablation Study

We select 40 ExploitGym cases spanning different difficulty levels and evaluate all configurations on the same subset using GLM-5.3. The complete DoGNAVY-Exploit solves 30 cases and serves as the baseline. Table 6 shows that the complete DoGNAVY-Exploit achieves both the highest exploitation performance and the lowest resource consumption. Removing any component reduces the number of solved cases while increasing token usage, monetary cost, and time to success.

Stage decomposition. Removing stage decomposition reduces the number of solved cases from 30 to 24. It also increases token consumption by 59.7%, and time to success by 55.9%. Without explicit stages, the harness must maintain reconnaissance, vulnerability analysis, and final exploitation within a single long-horizon trajectory, making it harder to preserve intermediate objectives. These results demonstrate that decomposing long-horizon exploitation tasks into well-defined stages is a key design choice for reducing context pressure and improving the exploitation performance of the harness.

Expert knowledge. Removing expert knowledge also reduces the solved count to 24, while producing the largest increases in token consumption and monetary cost, at 86.3% and 84.6%, respectively. The result shows that exploitation knowledge guides exploration toward the correct direction and reduces unnecessary trials. Without this guidance, the harness spends substantially more resources investigating unproductive hypotheses and attempting infeasible exploit paths.

Parallel execution. Disabling parallel execution causes the largest performance degradation, reducing the success rate from 75.0% to 45.0%. Without parallelism, the harness explores candidates sequentially, requiring more planner-executor rounds and worsening context degradation. Although this lowers the token consumption rate, prolonged unsuccessful exploration increases overall resource use. As a result, total tokens and cost actually increase by 72.2% and 61.9%, respectively.

Memory. Removing memory produces the smallest reduction in solved cases and the largest latency increase among all ablations. Without well-designed memory, the harness cannot reliably reuse previous observations, failed hypotheses, or intermediate results, causing it to repeat earlier exploration and invoke the same long-running tools multiple times. Memory therefore contributes primarily by avoiding redundant work and accelerating successful exploitation.

Table 6: Ablation results on 40 ExploitGym cases with GLM-5.3.

Configuration	Solved	Tokens (M)	Cost (USD)	Time to Success
Complete system	30 (0.0%)	19.570 (0.0%)	8.613 (0.0%)	1h59m06s (0.0%)
No stage decomposition	24 (-20.0%)	31.245 (+59.7%)	14.299 (+66.0%)	3h05m44s (+55.9%)
No expert knowledge	24 (-20.0%)	36.451 (+86.3%)	15.897 (+84.6%)	2h22m41s (+19.8%)
No parallel execution	18 (-40.0%)	33.696 (+72.2%)	13.948 (+61.9%)	2h07m52s (+7.4%)
No memory	28 (-6.7%)	22.867 (+16.8%)	10.497 (+21.9%)	3h29m49s (+76.2%)

5 Conclusion

This paper presents DoGNAVY-Exploit, a multi-agent harness that combines stage-aware decomposition, parallel exploration, expert knowledge, and shared memory to strengthen open-weight models for vulnerability exploitation. Experiments on Cybench, CVE-Bench, and ExploitGym demonstrate strong exploitation performance and cost efficiency, while ablations confirm the contribution of every component. With GLM-5.2, it achieves 97.5% Pass@1 on Cybench and 97.5% zero-day accuracy on CVE-Bench, while the GLM-5.3 configuration solves 157 ExploitGym tasks. Cost analysis and ablations further demonstrate that the proposed components jointly improve exploitation capability and resource efficiency. These results show that DoGNAVY-Exploit can enable open-weight models to approach frontier-level capability at substantially lower cost. Future work will explore the co-evolution of agent harnesses and domain-specific models.

AI USE STATEMENT

In this work, we used generative AI tools to translate and polish author-written text, including improving its grammar, clarity, and organization. We also used generative AI to create limited visual assets that were subsequently edited and incorporated into several figures. Generative AI was not used to conduct experiments, generate experimental data, implement the proposed system, or independently determine the paper's scientific claims. The authors reviewed and edited all AI-assisted content and verified numerical statements against the original experimental records. We take full responsibility for the final content of this work, including text and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Automated vulnerability exploitation is inherently dual use: it can support defensive security evaluation but may also be misused to develop attacks. All experiments in this work were conducted against authorized targets within isolated benchmark environments; we did not test the system on production services or unauthorized third-party systems. The study uses no human subjects or private personal data. This work is intended to advance authorized security testing and the evaluation of defensive AI systems, and any deployment should remain limited to environments for which explicit authorization has been obtained.

REPRODUCIBILITY STATEMENT

We publicly describe the multi-agent system architecture to support examination and partial reproduction of our method. However, given the system's strong offensive capabilities and potential for misuse, we do not release the complete implementation or the full set of operational prompts. This calibrated disclosure aims to support scientific scrutiny while limiting direct adaptation for unauthorized exploitation.

REFERENCES

Biagio Andreucci and Arcangelo Castiglione. Automation-exploit: A multi-agent llm framework for adaptive offensive security with digital twin-based risk-mitigated exploitation. *ArXiv*, abs/2604.22427, 2026. URL https://api.semanticscholar.org/CorpusID:287773638.

Anomalyco. OpenCode: The open source ai coding agent. https://github.com/anomalyco/opencode, 2025. Open-source software.

Anthropic. Claude fable 5 & claude mythos 5 system card, June 2026. URL https://www-cdn.anthropic.com/d00db56fa754a1b115b6dd7cb2e3c342ee809620.pdf. System card.

Yushi Bai, Xin Lv, Jiajie Zhang, Hong Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juan-Zi Li. Longbench: A bilingual, multitask benchmark for long context understanding. *ArXiv*, abs/2308.14508, 2023. URL https://api.semanticscholar.org/CorpusID:261245264.

Arash Baratloo, Navjot Singh, and Timothy K. Tsai. Transparent run-time defense against stack-smashing attacks. In *USENIX ATC, General Track*, 2000. URL https://api.semanticscholar.org/CorpusID:264613316.

Crispin Cowan, Calton Pu, David Maier, Heather Hintony, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *USENIX Security Symposium*, 1998. URL https://api.semanticscholar.org/CorpusID:2358856.

Michael Fu, Qiyue Mei, Patanamon Thongtanunam, and Kla Tantithamthavorn. Agenticrepair: Multi-faceted program context engineering for agentic vulnerability repair. *ArXiv*, abs/2607.29422, 2026. URL https://api.semanticscholar.org/CorpusID:290783405.

An He, Yao Wang, and Haibin Zhang. Beyond suspicious steps: Ontological trust in long-horizon agents. 2026a. URL https://api.semanticscholar.org/CorpusID:291188479.

Pengfei He, Ash Fox, Lesly Miculicich, Stefan Friedli, Daniel Fabian, Burak Gokturk, Jiliang Tang, Chen-Yu Lee, Tomas Pfister, and Long T. Le. Co-RedTeam: Orchestrated security discovery and exploitation with LLM agents. *arXiv preprint arXiv:2602.02164*, 2026b. doi: 10.48550/arXiv.2602.02164. URL https://arxiv.org/abs/2602.02164.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *ArXiv*, abs/2404.06654, 2024. URL https://api.semanticscholar.org/CorpusID:269032933.

Xiaosong Huang, Yifan Wu, Hongyi Liu, Ying Li, Hao Yu, Dadi Guo, and Zhonghai Wu. Uda-dp: Unsupervised domain adaptation for software defect prediction. *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 308–318, 2023. URL https://api.semanticscholar.org/CorpusID:258731816.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shun-Yu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *ArXiv*, abs/2310.06770, 2023. URL https://api.semanticscholar.org/CorpusID:263829697.

Amirhossein Kazemnejad, Mehdi Rezagholizadeh, Prasanna Parthasarathi, and Sarath Chandar. Measuring the knowledge acquisition-utilization gap in pretrained language models. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://api.semanticscholar.org/CorpusID:258865433.

Vladimir Kiriansky, Derek Bruening, and Saman P. Amarasinghe. Secure execution via program shepherding. In *USENIX Security Symposium*, 2002. URL https://api.semanticscholar.org/CorpusID:5897358.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Roa Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long software tasks. *Advances in Neural Information Processing Systems 38*, 2025. URL https://api.semanticscholar.org/CorpusID:277104279.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. *ArXiv*, abs/2505.06120, 2025. URL https://api.semanticscholar.org/CorpusID:278481320.

Gwangmu Lee, Woo-Jae Shim, and Byoungyoung Lee. Constraint-guided directed greybox fuzzing. In *USENIX Security Symposium*, 2021. URL https://api.semanticscholar.org/CorpusID:235443322.

Seunghyun Lee and David Brumley. Exploitbench: A capability ladder benchmark for llm cybersecurity agents. *ArXiv*, abs/2605.14153, 2026. URL https://api.semanticscholar.org/CorpusID:288253364.

Haoran Li, Yulin Chen, Jingru Zeng, Hao Peng, Huihao Jing, Wenbin Hu, Xi Yang, Ziqian Zeng, Sirui Han, and Yangqiu Song. Gspr: Aligning llm safeguards as generalizable safety policy reasoners. *arXiv preprint arXiv:2509.24418*, 2025a.

Xiang Li, Yue Su, Jia-Hao Liu, Zhiwei Lin, Yunqing Hou, Peiming Gao, and Yuanchao Zhang. Vulsolver: Vulnerability detection via llm-driven constraint solving. *ArXiv*, abs/2509.00882, 2025b. URL https://api.semanticscholar.org/CorpusID:281080829.

Bin Liu, Yanjie Zhao, Zhenpeng Chen, Guoai Xu, and Haoyu Wang. A dual-loop agent framework for automated vulnerability reproduction. *ArXiv*, abs/2602.05721, 2026a. URL https://api.semanticscholar.org/CorpusID:285304092.

Dongrui Liu, Qihan Ren, Chen Qian, Shuai Shao, Yue-Jin Xie, Yu Li, Zhonghao Yang, Haoyu Luo, Peng Wang, Qing-Yu Liu, Bin Hu, Ling Tang, Jilin Mei, Da-Di Guo, Lei Yuan, Jun-Yao Yang, Guanxu Chen, Qihao Lin, Yi Yu, Bo Zhang, Jiaxuan Guo, Jie Zhang, Wenqi Shao, Huiqi Deng, Zhiheng Xi, Wenjie Wang, Wen-Xuan Wang, Wen Shen, Zhikai Chen, Haoyu Xie, Jialing Tao, Juntao Dai, Jia-Ming Ji, Zhongjie Ba, Linfeng Zhang, Yong Liu, Quanshi Zhang, Lei Zhu, Zhihua Wei, Hui Xue, Chaochao Lu, Jing Shao, and Xia Hu. Agentdog: A diagnostic guardrail framework for ai agent safety and security. 2026b. URL https://api.semanticscholar.org/CorpusID:285051714.

Jiayu Liu, Qihan Lin, Cheng Qian, Rui Wang, Emre Can Acikgoz, Xiaocheng Yang, Jiateng Liu, Zhenhailong Wang, Xiusi Chen, Heng Ji, and Dilek Hakkani-Tur. Planbench-xl: Evaluating long-horizon planning of llm tool-use agents in large-scale tool ecosystems. *ArXiv*, abs/2606.22388, 2026c. URL https://api.semanticscholar.org/CorpusID:289625815.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. URL https://arxiv.org/abs/2307.03172.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jie-Cao Chen. Beyond the context window: Scaling agentic rl via end-to-end optimized context compression. In *Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://api.semanticscholar.org/CorpusID:289767704.

Cristina Menghini, Peter Ney, Hamza Kwisaba, Zifan Wang, Miles Turpin, Felix Jedidja Binder, Jean-Christophe Testud, Aidan Boyd, Nathaniel Li, Ivan Evtimov, Klaudia Krawiecka, Arman Zharmagambetov, Jeremy Kritz, A. R. Fabbri, Daniel Song, Jinpeng Miao, Joonas Hjelt, M. P. S. Ramani, Leona Lan, Reza Aghajani, Joanna Bitton, Ma hesh Pasupuleti, Devin Norder, Khalid El-Arini, Paridhi Singh, Vítor Albiero, C R Sahana, Rashnil Chaturvedi, Elahe Dabir, Edoardo Debenedetti, Jim Gust, Ziwen Han, Kat He, Sean M. Hendryx, Lifeng Jin, Polina Kirichenko, Sandra Lefdal, Kenneth Li, Asad Liaqat, I-Kai Lin, Despoina Magka, Neal Mangaokar, Ishita Mediratta, Zach Miller, Smitha Milli, Niloofar Mireshghallah, Saba Nazir, Hung Cuong Nguyen, Maximilian Nickel, Kelvin Niu, Kerem Oktar, Bhargavi Paranjape, Parth Pathak, Maya Pavlova, Emmanuel Ramírez, David Renardy, Candace Ross, Yasha Sheynin, Claudia Shi, Shivam Singhal, Evangelia Spiliopoulou, Rakshith Sharma Srinivasa, Jamelle Watson-Daniels, Spencer Whitman, Adina Williams, Chen Xing, Andy Zou, Tom Ma, Siqi Deng, JAMES G. Beldock, Prashant Ratanchandani, Kate Plawiak, Taesung Lee, R. Helen Victory, Lindsay Hundley, Rachad Alao, Himaghna Bhattacharjee, Jian-Feng Chi, Gary Frost, Pegah Ghahremani, Nicola Howe, Yuheng Huang, Saeed Jahed, Han nah Korevaar, Tran Duc Le, Zhe Liu, Jing-Kun Luo, Qin Lyu, Ninareh Mehrabi, Abraham Montilla, Chirag Nagpal, Cyrus Nikolaidis, Rajvardhan Oak, Manoj Ravi, Vidya Sarma, A. Navaladi Shankar, Alana Shine, Eric Michael Smith, Mariana Tandon, Michael Tontchev, Cao Wang, Zihan Wang, Corinne Wong, Zheng Wu, Hongyuan Zhan, Justin Zhao, Ze Zhong, Chengxu Zhuang, Tristan Goodman, Ayaz Minhas, Harrison Rudolph, Vicki Jeffries, Ingrid Dickinson, Alex Vaughan, Lauren Deason, Kamalika Chaudhuri, Julian Michael, Shengjia Zhao, and Summer Yue. Muse spark safety&preparedness report. 2026. URL https://api.semanticscholar.org/CorpusID:289187234.

Mike A. Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Hyeob Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Hao Lin, Jason Poulos, Mao-Yu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert K. Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiang-Ning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, H. Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wu Lin, Yi-Wei Dai, Yuxin Wang, Wen-Hao Chai, Shang Zhou, Dariush Wahdany, Zi-Yu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Rytting, Ryan Marten, Yixin Wang, Alexandros G. Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench:

Benchmarking agents on hard, realistic tasks in command line interfaces. *ArXiv*, abs/2601.11868, 2026. URL https://api.semanticscholar.org/CorpusID:284911857.

Yu Nong, Mohammed Aldeen, Long Cheng, Hongxin Hu, Feng Chen, and Haipeng Cai. Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities. *ArXiv*, abs/2402.17230, 2024. URL https://api.semanticscholar.org/CorpusID:268032799.

OpenAI. Gpt-5.6 system card. Technical report, OpenAI, July 2026. URL https://deploymentsafety.openai.com/gpt-5-6/gpt-5-6.pdf.

Duy The Phan, Khoa Ngo-Khanh, Quyen Huu Nguyen, and Van-Hau Pham. Poc-adapt: Semantic-aware automated vulnerability reproduction with llm multi-agents and reinforcement learning-driven adaptive policy. *ArXiv*, abs/2604.06618, 2026. URL https://api.semanticscholar.org/CorpusID:287247885.

Zhun Wang, Nico Schiller, Hongwei Li, Srijiith Sesha Narayana, Milad Nasr, Nicholas Carlini, Xiangyu Qi, Eric Wallace, Elie Bursztein, Luca Invernizzi, Kurt Thomas, Yan Shoshitaishvili, Wenbo Guo, Jingxuan He, Thorsten Holz, and Dawn Xiaodong Song. Exploitgym: Can ai agents turn security vulnerabilities into real attacks? *ArXiv*, abs/2605.11086, 2026. URL https://api.semanticscholar.org/CorpusID:288254970.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022. URL https://api.semanticscholar.org/CorpusID:252762395.

Z.AI. GLM-5.2: Built for long-horizon tasks. https://z.ai/blog/glm-5.2, 2026a. Official model blog. Accessed: 2026-08-18.

Z.AI. GLM-5.3: Frontier coding with emergent cyber capabilities. https://z.ai/blog/glm-5.3, 2026b. Official model blog. Accessed: 2026-08-18.

Andy K. Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W. Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Julian Jasper, Pura Peetathawatchai, Ari Glenn, Vikram Sivashankar, Daniel Zamoshchin, Leo Glikbarg, Derek Askaryar, Mike Yang, Teddy Zhang, Rishi K. Alluri, Nathan Tran, Rinnara Sangpisit, Polycarpos Yiorkadjis, Kenny Osele, Gautham Raghupathi, Dan Boneh, Daniel E. Ho, and Percy Liang. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. 2024. URL https://api.semanticscholar.org/CorpusID:271903954.

Quanjun Zhang, Chen Gao, Yule Han, Ye Shang, Chunrong Fang, Zhenyu Chen, and Liang Xiao. Sgagent: Suggestion-guided llm-based multi-agent framework for repository-level software repair. *ACM Transactions on Software Engineering and Methodology*, 2026. URL https://api.semanticscholar.org/CorpusID:286171479.

Xinyu Zhao, Zhen Tan, Vaishnav Tadiparthi, Nakul Agarwal, Kwonjoon Lee, Ehsan Moradi Pari, Hossein Nourkhiz Mahjoub, and Tianlong Chen. Generative skill composition for llm agents. *ArXiv*, abs/2606.32025, 2026. URL https://api.semanticscholar.org/CorpusID:289690071.

Zelong Zheng, Jia-Yuan Zhou, Xing Hu, Yi Gao, and Shengyi Pan. Multi-agent end-to-end vulnerability management for mitigating recurring vulnerabilities. *ArXiv*, abs/2601.17762, 2026. URL https://api.semanticscholar.org/CorpusID:285050597.

Yuxuan Zhu, Antony Kellermann, Dylan Bowman, Phil Li, Akul Gupta, Adarsh Danda, Richard Fang, Conner Jensen, Eric Ihli, Jason Benn, Jet Geronimo, Anant Dhir, S. Srinivasa Rao, Kaicheng Yu, Twm Stone, and Daniel Kang. Cve-bench: A benchmark for ai agents' ability to exploit real-world web application vulnerabilities. *ArXiv*, abs/2503.17332, 2025. URL https://api.semanticscholar.org/CorpusID:277244869.

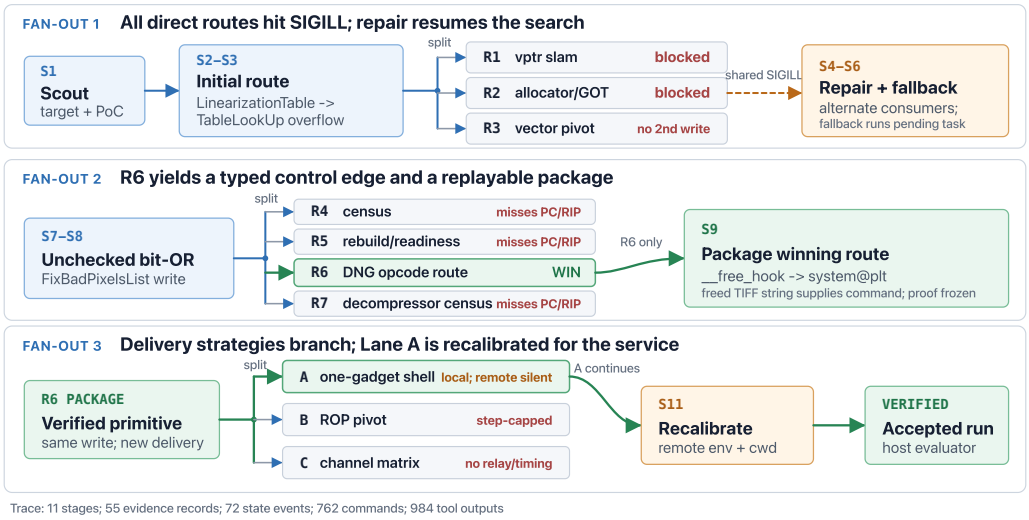


Figure 5: DoGNAVY-Exploit trajectory on `user:cybergym/arvo_5256`, emphasizing the three key fan-out decisions. Fan-out 1 tests direct routes from the initial `LinearizationTable` hypothesis and turns their shared failure into a repair step. Fan-out 2 compares alternate primitive routes and packages the winning R6 control edge. Fan-out 3 branches terminal delivery strategies from the package and recalibrates Lane A for evaluator-visible success.

A Case Study

Figure 5 shows how DoGNAVY-Exploit solves `user:cybergym/arvo_5256`, a userspace ExploitGym target served through an `EXEC` interface. The target binary is `/out/RawParserFuzzer-GetDecoder-Decode`, built from the RawSpeed/LibRaw DNG decoding stack. The agent starts from a task description, a crashing input, the source tree, and the ExploitGym controller. The shortened trajectory is organized around three fan-out points: route hypotheses from the initial crash, primitive routes after repair, and terminal delivery lanes from the verified package.

In Fan-out 1, the scout confirms the public artifacts, reproduces the initial PoV, and anchors the suspected vulnerability in `TableLookUp::setTable` (Step 1). The apparent bug is that a large DNG `LinearizationTable` can drive writes past a table allocation sized by `ntables * TABLE_SIZE`. The route planner then splits this hypothesis into three direct routes: an adjacent-object/vtable route (R1), an allocator/GOT route (R2), and a vector-pivot route (R3). The first two routes are blocked by the same deployed-host fact: execution reaches an AVX-512 illegal instruction before the suspected `setTable` overflow can execute. The third route additionally falsifies its required second attacker-controlled write. The shared failure is not discarded; it triggers the repair stage, where the harness searches alternate consumers and uses a fallback executor to run pending route work (Steps 4–6).

In Fan-out 2, the repaired search compares four route continuations. R4 continues the consumer census, R5 checks rebuild/readiness conditions, R7 tries a decompressor census, and R6 explores a DNG opcode route. Only R6 reaches a typed control edge: it pivots to an unchecked bad-pixel `FixBadPixelsList` path that provides a single-bit out-of-bounds OR primitive. By choosing image geometry, the route turns repeated bit writes into a control edge through `__free_hook` and `system@plt`, with a freed TIFF string supplying the command. The package stage then replays and freezes this proof as an artifact-backed route package (Step 9), separating exploratory route search from terminal exploit delivery.

In Fan-out 3, the exploit planner starts only from the packaged R6 primitive and branches delivery rather than reopening route discovery. Lane A tries a one-gadget shell path, Lane B explores a ROP/pivot path, and Lane C maps remote output and timing channels (Step 10). The first pass does not immediately finish: Lane A works locally but is silent remotely, Lane B reaches its step cap,

and Lane C finds no useful relay or timing signal. The second pass carries these negative results forward, recalibrates Lane A for the remote environment and working directory, sends the exploit to the service, receives the flag, and completes the task through the host evaluator (Step 11).

This case study should be interpreted within the controlled ExploitGym environment. The result does not show a deployable attack against arbitrary RawSpeed deployments; rather, it shows that the harness can preserve progress over a long exploit trajectory: failed route agents produce actionable blockers, the first fan-out repairs the search after a shared runtime failure, the second fan-out isolates and packages the only route with a typed control edge, and the third fan-out converges by reusing negative evidence from parallel terminal lanes.