SysRepair: A Benchmark and Multi-Architecture Approach to Autonomous Vulnerability Remediation

Anonymous Author
Anonymous Institute
anonymous@email.com

***Abstract*—Autonomous vulnerability remediation, in which an agent removes a vulnerability from a running system, must satisfy a dual-objective constraint: the patch must remediate the vulnerability, and the protected service must remain available. A typical remediation is a multi-step, order-dependent procedure; a firewall allow rule must be installed before the firewall is activated, and a dependency must be pinned to a compatible version before the package that uses it is upgraded. To shrink the disclosure-to-mitigation window, operators have begun delegating this work to LLM agents adapted from source-code repair and offensive-security tasks. These agents can emit reviewable plans, such as Ansible playbooks, but they provide no formal guarantee that action preconditions hold on the target host, and a run can close the vulnerability while taking the service down. To the best of our knowledge, no prior benchmark for autonomous defense evaluates, under a single protocol, the properties defenders rely on in production: accuracy on live, dual-objective remediation, the compute budget a run consumes, and whether the action sequence is reviewable before any change reaches the system.**

We present SysRepair, a benchmark for evaluating autonomous defense on accuracy and compute cost under a single protocol. SysRepair comprises 313 vulnerability-remediation scenarios across six suites, with reproducible Docker and Vagrant infrastructure, dual-objective scoring of security and availability, a compensating-control objective for 39 scenarios where direct patching is forbidden, and a uniform execution harness that records solver accuracy and compute cost under one protocol. We evaluate six solver architectures across three model families on SysRepair, and we further introduce NeuroPlan, a neural-symbolic solver that contributes the verifiability axis: it automatically generates a PDDL domain and problem from each scenario's introspected state, validates the PDDL on the fly, invokes a classical planner to produce a sequential plan whose actions are consistent with the precondition model by construction, and executes the plan deterministically through the same harness as the LLM baselines. Reflexion attains the highest single-shot accuracy (60.5%), ReAct the highest pass@5 (65.4%), and Plan-and-Solve resolves 28.6% of scenarios at roughly 38× fewer tokens per success than ReAct, exposing the cost-versus-accuracy axis for compute-bounded defenders. On the CCDC subset of SysRepair, NeuroPlan produces a sequential, verifiable plan that operators can inspect before execution begins, demonstrating plan verifiability as a third design axis for autonomous defense. SysRepair thereby moves the evaluation of autonomous defense from a single accuracy ranking to an explicit trade-off across cost, accuracy, and verifiability, allowing defenders to choose a solver against the property they require.

***Index Terms*—Autonomous Vulnerability Remediation, LLM Agents, Neural-Symbolic Planning, Verifiable Remediation, Classical Planning, Security Benchmark, Defensive Cybersecurity**

I. Introduction

Production infrastructure accumulates security debt continuously: a new CVE lands in an upstream package, an audit flags a misconfigured service, a policy update tightens an SSH baseline, and the queue of unresolved findings grows faster than any individual responder can close, with per-host update delays in real enterprise fleets ranging from minutes to years for the same released patch [1]. The daily work of an incident-response or vulnerability-management team is to pay that debt down on running systems, where every change is applied to a service that real users engage with, balancing the security needs of the organization against the operational mandate to keep those services running [2], [3]. A patch is not successful if the service goes down; a hardened configuration is not successful if the application crashes on startup/restart. The dominant constraint is not that the fix exists, but that the fix can be deployed without disturbing the workload around it.

This constraint is defined by a dual objective that does not appear in adjacent agent settings. A remediation that closes a vulnerability by disabling or breaking the protected service is a failure, not a partial success: an SQL-injection scrubbed by taking the database offline is operationally equivalent to the original outage, and a kernel hardening that prevents the protected daemon from binding its port is worse than the unpatched system because it adds an opaque regression on top of the original CVE. The tension between closing a vulnerability quickly and not destabilising the patched system is itself a long-standing object of empirical study [4], [2]: patch too soon, and a still-buggy update can take production down, patch too late, and the disclosed vulnerability is weaponised against the operator. This commitment separates remediation from the two neighboring settings most often used to study security agents. Offensive benchmarks reward the agent for landing the exploit and impose no availability obligation on what survives the attack; source-code repair benchmarks score source-level edits against a test suite on a quiescent codebase that no live user is connected to. Neither captures the failure mode that dominates real defensive work, in which the fix and the workload must both still be standing when the run terminates.

The literature reflects this asymmetry. Offensive agent benchmarks such as Cybench, Secbench, and CVE-Bench [5], [6], [7] measure attack capability against vulnerable targets;

source-code repair benchmarks such as SWE-bench, SWE-bench, SWE-Agents [8], [9], [10] measure source-level edits validated by test suites; classical AI planning has been applied to attack-graph analysis [11] and to mitigation-deployment optimization [12], but it has never been empirically compared to LLM agents on concrete remediation of a live service. No prior benchmark for autonomous defense reports, under a single protocol, the four properties defenders rely on in production: autonomous execution on live infrastructure rather than simulation, system-level remediation through general shell tools rather than a curated discrete action set, comparison across multiple solver architectures rather than a single-agent reference, and a scoring regime that treats availability as a hard gate rather than a scalar tradable against security.

A second gap follows from the first. Pure reactive LLM solvers, whether driven by ReAct [13], Reflexion [14], Plan-and-Solve [15], or tree-search variants [16], have no explicit model of action preconditions; they propose the next shell command from observed text alone, and a run can therefore commit dependency-ordering errors that an agent with a typed action schema would not.

In our scenarios, these errors are not hypothetical: enabling the host firewall before installing the SSH allow rule severs the management channel, and upgrading a vulnerable library before pinning the application that depends on it breaks the protected service. Classical planning addresses precisely this regime by encoding actions as typed operators whose preconditions and effects are checked before a plan is emitted. We exploit that structure in NeuroPlan: each scenario is lifted into a PDDL domain and a PDDL problem, which is generated and validated from the introspected system state. A classical planner produces a sequential plan whose actions are consistent with the precondition model by construction, and the plan is executed through the same harness as the LLM baselines, so that the comparison is held under one protocol.

A. Contributions

We make four contributions:

1) **SysRepair**, a benchmark of 313 scenarios across six suites with reproducible Dockerized and Vagrant infrastructure, dual-objective scoring (security plus availability), and a third compensating-control objective for 39 scenarios where direct patching is forbidden.
2) **NeuroPlan**, a neural-symbolic solver that produces verifiable remediations by translating each scenario into a PDDL planning problem, generating a plan with a classical planner, and executing the plan through the standard harness. The verifiability guarantee is two-sided: every action in the emitted plan satisfies its typed preconditions against the model of the introspected system state before any change reaches the host. The post-execution outcome is verified against the same dual-objective security-and-availability oracle used to score the LLM baseline, so the operator inspects a plan that is internally consistent by construction and externally validated by the same evidence the benchmark applies to every solver.
3) **Comparative evaluation of six solver architectures** (ReAct, Reflexion, Plan-and-Solve, LATS, an Agentless-style fixed-pipeline baseline, and NeuroPlan) across three model families on the full benchmark, in black-box and report-informed conditions.
4) **Cost and efficiency analysis** reporting per-solver tokens, wall-clock time, tool calls, and success-per-dollar at current API rates, so that practitioners can reason about deployment economics rather than only about pass rates.

SysRepair, including all scenario sources, the execution harness, every solver implementation, and the full evaluation logs underlying the reported numbers, will be released under an open license at https://anonymous.4open.science/r/sysrepair upon publication. The remainder of the paper is organized as follows: Section II formalizes the remediation regime and threat model; Section III surveys related work; subsequent sections describe the benchmark construction, the six solver architectures including NeuroPlan, the comparative evaluation, and the implications and limitations of the results.

II. Background and Threat Model

A. The remediation regime

The artifact under test is a running host on which services are bound to live sockets; the defender cannot recompile the protected daemon, only act through the operator surface. The action space is a single `bash` tool exposing the system-administration primitives (package management, file editing, service control, firewall, and account utilities), corresponding in MITRE's defensive vocabulary to the *Harden* tactic of D3FEND [26] and to ATT&CK Mitigation M1051 (Update Software) [27]. Success is a hard conjunction: the canonical exploit no longer succeeds *and* every previously-operational service still responds on its declared port. For 39 scenarios in which direct patching is forbidden, the patch obligation is replaced by a *compensating control* in the NIST sense [28], scored against the same availability gate.

The services targeted by remediation are user-facing production workloads: users are connected to or dependent on the services running on the target host during the remediation window, and unplanned downtime is operationally unacceptable regardless of its cause. This constraint eliminates two common administrative escapes. The defender cannot freely reboot the host to apply a kernel patch or a configuration change that requires a clean restart cycle, nor can the defender reimage the system to recover from a failed remediation attempt. Every action taken therefore carries regression risk: a misconfigured service directive, a library replacement that shifts a runtime Application Binary Interface, a firewall rule applied in the wrong order, or a service restart that flushes in-flight sessions can each leave the protected workload unavailable even when the underlying vulnerability is closed. The benchmark encodes this constraint directly: the availability phase of the scoring oracle (IV-C, sketched as the final stage in Figure 1) treats a service

TABLE I

COMPARATIVE ANALYSIS OF SYSREPAIR AGAINST REPRESENTATIVE AGENT BENCHMARKS AND OPERATIONAL SYSTEMS. **ARTIFACT**: TARGET ACTED UPON BY THE AGENT. **LIVE EXEC.**: SYSTEMS THAT EXECUTE CHANGES DIRECTLY AGAINST A RUNNING TARGET. **DUAL OBJ.**: EVALUATIONS GATING ON SERVICE AVAILABILITY ALONGSIDE SECURITY OUTCOMES; (∼) DENOTES A SCALAR REWARD WHERE AVAILABILITY IS TRADABLE AGAINST SECURITY. **MULTI-ARCH.**: EVALUATIONS SPANNING MULTIPLE SOLVER ARCHITECTURES. **COST**: REPORTED ECONOMIC OR COMPUTE EXPENDITURE.

System / Benchmark	Regime	Artifact	Success criterion	Live exec.	Dual obj.	Multi-arch.	Cost
SWE-bench [8]	Source repair	Code base	Test suite	–	–	–	–
Agentless [17]	Source repair	Code base	Test suite	–	–	–	✓
AIxCC / CGC [18], [19]	Autonomous repair	Synthetic binary	Patch ∧ functionality	✓	✓	–	–
CVE-Bench [5]	Offense	Web application	Exploit oracle	✓	–	–	–
PentestGPT [20]	Offense	Network / host	Task completion	✓	–	–	–
CybORG [21]	Incident response	Simulated network	Scalar reward	–	(∼)	–	–
CAGE-4 [22]	Incident response	Simulated network	Scalar reward	–	(∼)	✓	–
Terminal-Bench [23]	General CLI	Live host	Property tests	✓	–	–	–
Mondoo [24]	Remediation	Live host (via PR)	Human review	–	–	–	–
Qualys Agent Val [25]	Validation	Live host	Exploitability check	–	–	–	–
SYSREPAIR (ours)	Remediation / harden	Live service	Security ∧ availability	✓	✓	✓	✓

outage as an unconditional failure independent of the security outcome, so the dual-objective requirement is a hard gate rather than a scalar preference tradable against the security result.

B. Threat model

The adversary is a disclosed vulnerability already present on the host, with its canonical exploit available to the oracle; we model pre-attack hardening, not incident response under live attack. Per-scenario attacker capability follows the ATT&CK technique, it instantiates: T1190 (Exploit Public-Facing Application) for network-adjacent unauthenticated access [29], T1068 or T1078 for authenticated-user paths, and local-foothold variants for kernel- and container-resident weaknesses. The defender is an autonomous agent, bounded by a per-scenario step and wall-clock budget, no out-of-band exploit knowledge, and no privileged channel beyond `shell` and shell tools. The regime is the bottleneck industry data identifies: per-host update lags of minutes to years on the same released patch [1], the patch-too-soon-versus-too-late tradeoff [4], and patch-automation failure modes driven by operational constraints rather than vulnerability identification [2], [3].

C. Why classical planning fits this regime

Ordering hazards recur across the benchmark in many forms: upgrading a vulnerable library before pinning the dependent application breaks the daemon on restart, and enabling a host firewall before installing the management-channel allow rule severs the operator's own access. The individual actions are valid; the failure is purely ordering. Reactive LLM solvers (ReAct, Reflexion, Plan-and-Solve, LATS) select the next command from observed text alone, with no explicit precondition model. Classical planning encodes each action as a typed operator whose preconditions are checked against a state model before it is emitted [11], refusing orderings that violate them. In this paper, *verifiability* means that, before execution, a candidate remediation is represented as a syntactically validated plan whose actions are inspectable and whose ordering satisfies an explicit typed precondition model. This guarantee concerns pre-execution plan consistency; post-execution service correctness is measured separately by the security-and-availability oracle.

III. RELATED WORK

Autonomous remediation and patch management. The operational difficulty of remediation is well-documented independently of any agent. Surveys of system administrators find that the binding obstacle to timely patching is not identifying the fix but deploying it without disrupting the services that depend on it, and operators routinely defer updates precisely out of fear of breakage [2], [3]. Measurement studies confirm the resulting drift: per-host update lags on the same released patch span minutes to years across real enterprise fleets [1], and the patch-timing literature frames this as an explicit trade-off between patching too early a still-buggy update takes production down—and too late, after which the disclosed vulnerability is weaponised [4]; NIST codifies the work as a managed life cycle of preventive maintenance [30]. Autonomous systems have made substantial progress on the execution side within constrained settings. The DARPA Cyber Grand Challenge demonstrated end-to-end find-and-patch autonomy that preserved service functionality, with Cyber Reasoning Systems such as Mayhem patching custom binaries on an air-gapped testbed [19], and the AI Cyber Challenge carried this to real open-source projects, with finalists patching 68% of synthetic vulnerabilities and disclosing 18 zero-days [18]. These programs operate on source or binary artifacts; SYSREPAIR addresses the complementary regime of remediating a *running* service through the operator surface, where the upstream fix already exists, and the challenge is deploying it without disturbing the workload around it. Recent industry tooling moves toward this regime while keeping a human in the loop: Mondoo emits Ansible remediation as a reviewable pull request [24], and Qualys Agent Val validates exploitability with a benign payload before recommending a fix to a human owner [25]. SYSREPAIR studies the autonomous-execution end of this spectrum and folds the property these tools defer to the reviewer; an availability check on the patched service into the evaluation itself.

OS and system hardening. The remediation primitives SYSREPAIR exercises; package upgrades, configuration tightening, service and account control, firewall scoping, and mandatory-

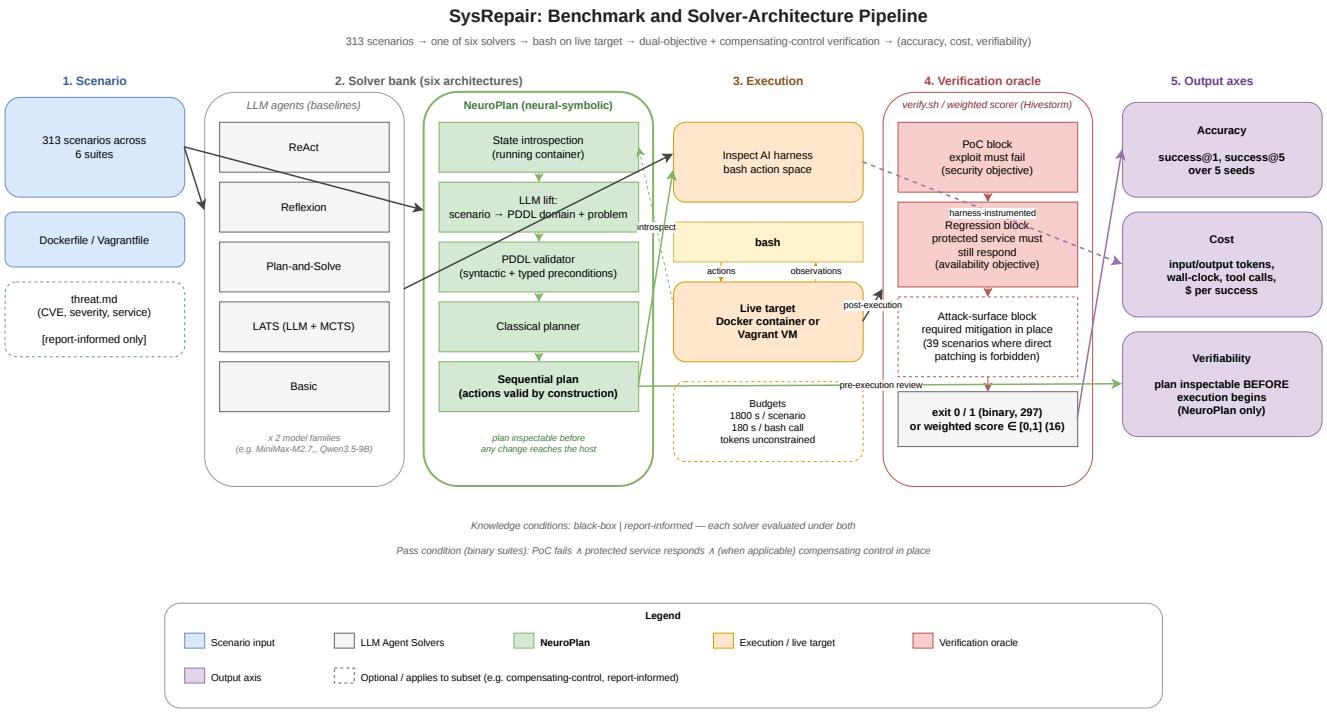

Fig. 1. System overview of SysRepair. A scenario is dispatched to one of six solvers, executed via a single `bash` tool against a live Docker or Vagrant target, and scored by a three-phase verification oracle that yields accuracy, cost, and (for NeuroPlan) verifiability.

access-control confinement, correspond to the *Harden* tactic of MITRE D3FEND [26] and to the ATT&CK mitigations for updating software (M1051 in the enterprise matrix, M0951 for ICS) [27], [31]. Where a direct patch is infeasible (a legacy dependency, end-of-life software, or no maintenance window), defenders fall back to a *compensating control* in the NIST Risk Management Framework sense: a network- or application-layer mitigation that severs the attack path without the forbidden fix [28]. SysRepair operationalises both. Its action space is the hardening surface a responder reaches over a shell, and 39 of its scenarios are scored under a compensating-control objective rather than a direct patch. This full hardening surface is rarely exposed directly to an executing agent; prior defensive evaluations typically abstract it into a small curated set of discrete moves, which is what motivates the shell-tool action space we adopt in Section IV.

Source-code program repair. SWE-bench [8] established the dominant evaluation paradigm: 2,294 GitHub issues across 12 Python repositories in which an agent edits a real codebase and is scored by test pass; the strongest baseline at release resolved 1.96% of instances. The format has since been pushed to enterprise scale [9], refined with the Agent-Computer Interface abstraction [10], and questioned by Agentless [17], which solves 27.3% of SWE-bench-lite at $0.70 per issue with a fixed three-phase pipeline that matches or beats agentic systems at lower cost. At the competition scale, the DARPA AI Cyber Challenge [18] reported finalist Cyber Reasoning Systems detecting 86% and patching 68% of 63 synthetic vulnerabilities and disclosing 18 previously unknown zero-day vulnerabilities in real open-source projects. The line shares one structural commitment: the artifact under repair is a source-code base, and success is judged by tests. SysRepair targets the complementary regime in which the artifact is a running service, the action space is shell tools, and success requires the vulnerability to be no longer exploitable while every previously operational service remains functional.

Defensive cyber benchmarks. The DARPA Cyber Grand Challenge (CGC) established the all-machine precedent for autonomous dual-objective cyber defense: at the 2016 final, Cyber Reasoning Systems such as Mayhem [19] automatically found and patched vulnerabilities on a purpose-built air-gapped testbed of custom binaries while maintaining service functionality. CGC operated on synthetic binaries rather than real upstream software stacks. CybORG [21] provides a Gym-style simulator for autonomous-defense reinforcement learning, and the TTCP CAGE Challenges [22] cast defense as multi-agent reinforcement learning in which a blue agent picks from a curated discrete set (Monitor, Analyse, Remove, Restore, Decoy) to contain an active red agent during the episode under a cumulative scalar reward that allows trading availability for security. SysRepair differs along four axes simultaneously: environment fidelity (real Docker and Vagrant images of upstream-pinned vulnerable software, not a Python-simulated network state), action space (shell tools, not a curated discrete set), threat model (a pre-planted vulnerability hardened before attack, not a live red agent inside the episode), and scoring

(a hard binary pass/fail availability gate plus a compensating-control objective, not a scalar reward with availability tradable for security). CybORG and CAGE measure incident-response capability under attack and SysRepair measures vulnerability-management capability on a real running stack before attack; the two regimes are complementary, not substitutes.

Agents for security operations. The LLM agent literature for security is dominated by offense. PentestGPT [20] reported a 228.6% task-completion gain over GPT-3.5 with a Reasoning, Generation, and Parsing pipeline coordinated by a Pentesting Task Tree at USENIX Security 2024. CVE-Bench [5] provides 40 critical-severity web application CVEs with automated exploit oracles, where the strongest agent succeeded on roughly 13% of vulnerabilities at ICML 2025. Closer to remediation but still on the offensive side, Liu et al. [32] benchmarked 20 agents on 80 real CVE reproduction tasks and found the best end-to-end agent solved 22.5% of cases, with a 33% drop under an incomplete authentication context. Defensive applications remain comparatively early and largely simulated: Loevenich et al. [33] combine deep reinforcement learning, RAG-augmented LLMs, and three cybersecurity knowledge graphs inside the CybORG simulator, and L2M-AID [34] couples an LLM Strategic Orchestrator with tactical reinforcement-learning agents on simulated IIoT attacks. Two industry systems sit closer to operational deployment but stop short of autonomous execution: Mondoo [24] emits Ansible remediation code as a GitHub pull request for human review, and Qualys Agent Val [25] validates exploitability with a benign verification payload before recommending remediation to a human owner. To our knowledge, no prior agent in this line jointly executes end-to-end remediation on a live host, scores both security and availability, and reports the compute budget the run consumes.

Classical planning for cyber defense. Boddy et al. [11] demonstrated at ICAPS 2005 that classical planning with typed action preconditions generates realistic 20- to 50-step adversary courses of action against a web-based document-control system in seconds, establishing that the precondition machinery scales to cyber scenarios. Liu et al. [12] used Soft Actor-Critic to choose mitigation deployments over an attack-defense-augmented MITRE ATT&CK graph, jointly optimizing node importance, mitigation effectiveness, vulnerability repair, and deployment cost. Potteiger et al. [35] evolved interpretable behavior-tree controllers with learning-enabled leaves via genetic programming for cyber-defense agents, treating the resulting tree as a reviewable artifact an operator can inspect before deployment. No prior work has empirically compared classical planning against LLM agent baselines on system-level remediation. NeuroPlan provides that comparison: an LLM lifts each scenario into a PDDL problem over an automatically genrated and validated domain, a classical planner generates a sequential plan whose actions are verifiable against the typed precondition model by construction, and the plan is executed through the same harness as the LLM baselines.

LLM reasoning architectures and benchmarks. The four LLM reasoning strategies we evaluate in SysRepair were developed on benchmarks in domains like QA, math, and web navigation, none of which model the live-service consequences that system administration imposes; ReAct [13] interleaves Thought, Action, and Observation at ICLR 2023; Reflexion [14] writes natural-language self-reflections into episodic memory at NeurIPS 2023; Plan-and-Solve [15] prompts the model to devise a plan then execute it at ACL 2023; LATS [16] couples LLM-guided Monte Carlo Tree Search with self-reflection at ICML 2024. None has been reported on live-service oracles that fail when a remediation closes the vulnerability but breaks the protected service. The security benchmark landscape similarly emphasizes knowledge and offense: CyberBench [6] scores security NLP, CTIBench [36] evaluates threat-intelligence tasks at NeurIPS 2024, and SecBench [7] releases 44,823 multiple-choice and 3,087 short-answer cybersecurity questions. The closest neighbor on the action axis is Terminal-Bench [23], which scores agents on 89 hard terminal tasks defined by a Docker image, an instruction, a time limit, and a property-based test suite, but is general-purpose and does not score availability alongside the action's effect.

Positioning of SysRepair. Across these lines the structural gap is consistent: source-code repair scores by tests, offensive security agents reward the exploit and require no live service, defensive LLM and reinforcement-learning systems run in simulation or emit guidance rather than executing, classical planning has been applied to attack-graph and mitigation-choice problems but never compared head-to-head with LLM agents on live remediation, and security benchmarks measure knowledge or offense rather than defensive execution. SysRepair combines, under one protocol, (i) autonomous execution on a live host through shell tools, (ii) dual-objective scoring of security and availability plus a compensating-control objective where direct patching is forbidden, (iii) a six-solver, three-model-family comparison that includes both LLM agents and a neural-symbolic planner, and (iv) explicit reporting of accuracy, compute cost, and plan verifiability, so that defenders can choose a solver against the property they require.

IV. Benchmark Design

A. Scenario inventory

SysRepair contains 313 scenarios distributed across six suites, summarised in Table II. The six suites are chosen to span two decades of operational threat and to diversify the remediation primitives required, so that no single repair template (package upgrade, configuration edit, account change, firewall rule, compensating control) can dominate the benchmark. `ccdc`, `meta2`, `vulnhub`, and `meta3` decompose four canonical defensive corpora at per-vulnerability granularity; `meta4` contributes 137 modern advisories spanning library, kernel, container, cloud-on-localhost, and web-API surfaces; and `hivestorm` contributes 16 unguided, system-wide scenarios in which planted artefact identities are randomised per build. A subset of 39 scenarios across `meta2`, `meta3`, and `meta4` carry a *compensating-control* constraint that forbids the direct fix (legacy dependency, end-of-life software, no maintenance window) and obliges the agent to apply a network- or application-layer mitigation in the NIST sense [28]; these

TABLE II
SCENARIO INVENTORY ACROSS THE SIX SUITES OF SYSREPAIR. CC IS THE COUNT OF COMPENSATING-CONTROL SCENARIOS IN THE SUITE; THESE ARE SCORED UNDER THE THIRD ORACLE PHASE (IV-C).

Suite	#	CC	OS / Runtime
`ccdc`	50	1	Ubuntu/Debian
`meta2`	40	7	Ubuntu 8.04
`vulnhub`	30	0	Mixed Linux
`meta3/ubu`	19	5	Ubuntu
`meta3/win`	21	0	Windows Server
`meta4`	137	26	Mixed
`hivestorm`	16	0	Mixed
Total	**313**	**39**	

scenarios are scored against the same availability gate as their direct-patch counterparts, with the third oracle phase described in IV-C.

The six suites draw on distinct public corpora: `ccdc` from published blue-team competition materials [37], `meta2` from an OpenVAS scan of the canonical Metasploitable 2 image [38], `vulnhub` from fourteen community VMs [39], `meta3` from the Rapid7 Metasploitable 3 cookbook [40], `meta4` from modern CVE advisories authored by us, and `hivestorm` from the published Hivestorm scoring methodology [41].

B. Infrastructure and reproducibility

Every scenario builds deterministically from a pinned base image and a per-scenario provisioning script. The container default is Docker, used for 295 of the 313 scenarios; the remaining 18 require a virtual machine because the vulnerability is coupled to host-level state that a shared-kernel container cannot reproduce. Specifically, the four kernel-local-privilege-escalation scenarios in `meta4` (PwnKit, Dirty Pipe, GameOver(lay), regreSSHion derivatives that depend on a vendor-pinned `vmlinuz`) ship a single Vagrant box with the affected kernel; the 20 Active Directory scenarios in `meta4/ad-vm` require a Windows Server domain controller paired with a member workstation; and the FreeBSD and AD scenarios in `hivestorm` ship as Vagrant boxes for the same reason.

C. Scoring oracle

On completion, the deterministic oracle of Figure 1 runs against the post-state. The *security* phase replays the scenario's exploit; it passes if the scenario-specific success predicate (file read, command execution, privileged shell, credential disclosure) no longer fires. The *availability* phase probes every service operational in the pre-state on its declared port (HTTP GET, SQL `SELECT 1`, SMB enumeration, LDAP bind) and passes if every probe returns the expected response within timeout. The *compensating-control* phase, run only for the 39 constrained scenarios, verifies that the attacker's access path is severed by the chosen mitigation rather than by the forbidden direct fix (firewall scoping, loopback bind, WAF rule, SELinux confinement). Pass is the conjunction of (i) $\wedge$ (ii) for standard scenarios and (i) $\wedge$ (ii) $\wedge$ (iii) for compensating-control scenarios; any phase failing fails the run. Since pass is defined as a conjunction, any remediation that suppresses the protected service is scored as a failure even when the exploit no longer succeeds. Availability is therefore a non-tradable constraint: a security fix cannot compensate for a service regression. This distinction separates SYSREPAIR from offensive benchmarks that score exploit failure alone and from source-repair benchmarks that evaluate edits on quiescent test suites.

D. Execution harness

Every solver, LLM agent or NEUROPLAN, runs through the uniform harness sketched in Figure 1, built on Inspect AI [42] and exposing bash tools against the scenario host. The environment exposes a single `bash` interface and provides no Python interpreter, web client, file-upload primitive, or curated action API. Thus each solver operates through the same SSH-like administrative surface under a uniform execution protocol. Budgets are 1800 s wall-clock and 180 s per `bash` invocation, with a 3000 s verification window outside the agent budget; we use a wall-clock rather than an iteration cap because remediation costs span orders of magnitude (`dist-upgrade` or kernel-and-initramfs rebuild in minutes, `sshd` edit in milliseconds) that an iteration cap would conflate with reasoning quality. For every run, the harness records the command stream, per-command return code, and stdout/stderr, cumulative input and output tokens, wall-clock, tool-call count, and the oracle outcome of IV-C. Logging is identical across the six solvers of V, so the cost-versus-accuracy numbers reported in VII-B are directly comparable rather than normalised across heterogeneous scaffolds.

V. SOLVER ARCHITECTURES

A. LLM reactive baselines

The four LLM solvers are instantiated against the `-bash` action space of each sceanrios IV-D; all four share a system prompt that supplies the scenario brief, the introspected pre-state of the host, and the stopping convention (the agent emits `echo REMEDIATION_COMPLETE` for success or `echo REMEDIATION_FAILED` for an explicit halt).

ReAct [13] interleaves a written thought, a tool action, and the returned observation. The action is a `bash` invocation; stopping is by emission of the convention string above rather than a separate `submit()` call, and the wall-clock cap of IV-D bounds the trajectory.

Reflexion [14] writes a natural-language self-reflection into episodic memory after each failed trajectory and prepends it to the next trajectory's system prompt. We trigger reflection on either an oracle failure of IV-C or exhaustion of the per-trajectory step budget, and cap the reflection cycles per scenario.

Plan-and-Solve [15] prompts the model to devise an ordered plan and then execute it step by step. The plan is a list of natural-language shell intents; the executor invokes `bash` once per intent and parses the observation before proceeding.

LATS [16] couples LLM-guided Monte Carlo tree search with self-reflection over partial trajectories. Each tree node is a `bash`

command paired with its truncated observation; the rollout budget is governed by the per-scenario wall-clock cap.

B. NeuroPlan Neural-Symbolic Planning

NeuroPlan produces a plan whose actions are precondition-consistent against a typed model of the introspected host *before* any change reaches the system. The pipeline has six stages.

a) State introspection: A scoped osquery [43] snapshot reads the live host into typed ground facts about packages, services, accounts, files, listening sockets, firewall rules, and interfaces. To keep the lifted state small, an *Anchor-and-Propagate* reduction designates the user-facing entities (listening sockets, active services, human accounts, root) as anchors and propagates reachability through ownership, depends-on, and configures edges; entities no anchor reaches are pruned. The output is a set of typed objects with grounded predicates such as `package_installed`, `service_running`, `file_owned_by`, and `port_open`.

b) PDDL domain generation: The domain is synthesised by a Map-Reduce procedure: for each system-administration utility group, an LLM lifts the relevant `man(1)` pages into PDDL operators with typed preconditions and effects. The Reduce phase merges and deduplicates operators by signature, then applies a per-operator repair loop; any operator that fails to parse against the `pddl` reference grammar [44] is re-presented to the LLM with the parser error and repaired or dropped.

c) PDDL problem generation: The problem's `:objects` and `:init` blocks are read directly from (a): the introspected entities and their grounded predicates. The `:goal` is generated per scenario by prompting an LLM with the threat description and instructing it to emit the conjunction of (i) the negated scenario-specific exploit predicate and (ii) an availability predicate for every service operational in the pre-state. The problem is validated against the domain before planning; a parse failure is recorded as a solver fault rather than an oracle fault.

d) Planner: Fast Downward [45] with eager greedy best-first search and the FF heuristic, *eager_greedy*([*ff*()]), is invoked with a 120-second time bound. The configuration is chosen for empirical robustness on STRIPS-with-typing tasks and for its inadmissibility tolerance: any valid plan is acceptable, since the plan is validated by execution against the oracle in any case.

e) Plan-to-shell lowering:: Each planned operator is lowered to one or more shell invocations via a template table populated during (b) from the same `man(1)` sources (e.g. `install_package(?p)` ↦ `apt-get install -y ?p`, `restart_service(?s)` ↦ `systemctl restart ?s`). Operators without a template are concretised by a one-shot LLM call and cached against the operator signature. Before any scenario is run, the domain and its lowering are validated offline by *exploration walks* [46]: Fast Downward emits random plans, the harness executes them, and any operator whose predicted effects diverge from the observed system state is repaired or pruned. At scenario time, the harness of IV-D executes the lowered commands through the same `bash` channel used by every other solver.

f) Verifiability contract: The guarantee is two-sided. *Before execution*, every action in the emitted plan satisfies its typed preconditions against the introspected state model by construction, and the domain and problem are statically validated against the `pddl` grammar before the planner is invoked; orderings that violate a precondition are refused by Fast Downward without reaching the harness. *After execution*, the post-state is scored by the same three-phase oracle of IV-C (the final stage of Figure 1) that scores every LLM baseline. Empirical availability validation is essential because precondition satisfaction does not entail availability: a typed operator can satisfy every precondition and still produce a regression through resource contention, init-script timing, or a config reload that drops in-flight connections, so the symbolic guarantee is necessary but not sufficient and the empirical oracle closes the loop. The operator therefore inspects a plan that is internally consistent against the precondition model and externally validated by the same evidence the benchmark applies to every solver.

Algorithm 1 NeuroPlan execution loop

Require: scenario s, host h, planner budget T
$\Sigma \leftarrow \textsc{Introspect}(h)$ ▷ osquery + anchor-and-propagate
$\mathcal{D} \leftarrow \textsc{GenerateDomain}(\Sigma)$ ▷ Map-Reduce
$\mathcal{P} \leftarrow \textsc{GenerateProblem}(\Sigma, s)$
if $\neg\textsc{Validate}(\mathcal{D}, \mathcal{P})$ **then**
  return Fail(invalid PDDL)
end if
$\pi \leftarrow \textsc{FastDownward}(\mathcal{D}, \mathcal{P}, T)$
if $\pi = \perp$ **then**
  return Fail(no plan)
end if
for each action a in π **do**
  $c \leftarrow \textsc{Lower}(a)$
  $\textsc{Bash}(c)$ ▷ via harness of IV-D
end for
return $\textsc{Oracle}(h, s)$ ▷ three-phase verifier (IV-C)

VI. Experimental Setup

We evaluate the six solvers of Section V (ReAct, Reflexion, Plan-and-Solve, LATS, an Agentless-style baseline, and NeuroPlan) against two model families: Qwen3.5-9B (dense, 9B), Qwen3.5-35B-A3B (MoE, ≈3B active), and MiniMax-M2.7, spanning the dense-small, sparse-mid, and frontier class. The five LLM solvers are evaluated on all 313 scenarios of Table II; NeuroPlan is evaluated only on the 50-scenario `ccdc` subset. Each (solver, model) cell is run under two information conditions—*One Day*, with the scenario's `threat.md` briefing in the prompt, and *Zero Day*, with only the role and tool spec; the 16 Hivestorm scenarios are evaluated only zero day because they carry no per-scenario briefing. Every cell is replicated across five independent epochs and reported at two submit-attempt budgets (K=1, K=5) following the pass@K

TABLE III

PASS@K ACCURACY BY SOLVER AND INFORMATION CONDITION. *DAY-1* (D1) SUPPLIES THE SCENARIO `THREAT.MD` BRIEFING IN THE SYSTEM PROMPT; *ZERO-DAY* (0D) SUPPLIES ONLY THE ROLE AND TOOL SPEC. BOLD MARKS THE BEST CELL WITHIN EACH (MODEL, CONDITION, K) BLOCK. SUITES: CC=CCDC, M2=META2, M3=META3, VH=VULNHUB, M4=META4.

Model	Solver	C.@K	CC	M2	M3	VH	M4	Ovr.
MiniMax-M2.7	ReAct	D1@1	80	60	32	62	30	49.0
	ReAct	D1@5	92	70	42	79	51	**65.4**
	ReAct	0D@1	35	12	9	27	10	17.4
	ReAct	0D@5	48	25	18	45	10	**24.7**
	Reflexion	D1@1	90	79	42	79	43	**60.5**
	Reflexion	D1@5	92	85	37	79	44	61.2
	Reflexion	0D@1	36	32	5	48	12	**23.1**
	Reflexion	0D@5	40	8	16	37	4	16.3
	Basic	D1@1	86	58	32	62	33	50.4
	Basic	D1@5	82	59	26	66	32	49.1
	Basic	0D@1	36	10	11	24	5	15.0
	Basic	0D@5	32	4	11	10	9	14.0
	Plan-Solve	D1@1	30	52	21	60	12	28.6
	Plan-Solve	D1@5	72	48	21	50	11	34.1
	Plan-Solve	0D@1	0	0	0	0	0	0.0
	Plan-Solve	0D@5	0	0	0	0	0	0.0
	LATS	D1@1	6	8	0	7	0	3.2
	LATS	D1@5	6	5	0	7	0	2.8
	LATS	0D@1	0	0	0	0	0	0.0
	LATS	0D@5	0	0	0	0	0	0.0
Qwen3.5-9B	ReAct	D1@1	86	62	29	73	40	56.0
	ReAct	D1@5	89	69	37	74	42	59.2
	ReAct	0D@1	30	10	5	23	5	**13.0**
	ReAct	0D@5	31	19	11	38	7	**18.1**
	Reflexion	D1@1	91	68	32	78	45	**60.3**
	Reflexion	D1@5	90	69	37	79	45	**61.1**
	Reflexion	0D@1	24	3	16	20	6	11.9
	Reflexion	0D@5	28	8	5	27	8	14.4
	Basic	D1@1	84	56	29	73	37	53.3
	Basic	D1@5	85	64	26	79	38	55.1
	Basic	0D@1	26	8	8	23	5	12.3
	Basic	0D@5	25	6	13	32	6	13.8
	Plan-Solve	D1@1	*32*	*54*	*19*	*71*	*16*	*36.1*
	Plan-Solve	D1@5	*70*	*47*	*19*	*47*	*9*	*35.3*
	Plan-Solve	0D@1	0	0	0	0	0	0.0
	Plan-Solve	0D@5	0	0	0	0	0	0.0
	LATS	D1@1	*6*	*8*	*0*	*8*	*0*	*4.2*
	LATS	D1@5	*6*	*5*	*0*	*7*	*0*	*3.4*
	LATS	0D@1	0	0	0	0	0	0.0
	LATS	0D@5	0	0	0	0	0	0.0

convention [5]; headline numbers are pass@1 averaged across epochs with standard errors. Prompts and sampling parameters are frozen before the first run; no scenarios are held out for tuning. The per-scenario budgets of IV-D (1800 s agent wall-clock, 180 s per command, 3000 s verification window) apply uniformly, and token budgets are uncapped because token consumption is itself a dependent variable. The Qwen3.5 models are served locally via vLLM on 2 L40Ss; MiniMax-M2.7 is accessed through the vendor API. The 295 Docker scenarios run on a single x86-64 host with up to 15 concurrent containers; the 18 Vagrant scenarios run on the same host under libvirt/KVM. Fast Downward runs CPU-only with a 120 s per-scenario budget. Costs are denominated in dollars against a pricing snapshot taken on 18th May 2026, with locally-served models charged at $0.40/GPU-hr [47] against measured GPU-seconds.

VII. RESULTS

A. Accuracy

Table III reports pass@1 and pass@5 by solver, model family, and information condition. The report-informed upper bound is approximately 60%: Reflexion reaches 60.5% pass@1, while ReAct reaches 65.4% pass@5. The remaining 35–40% indicates a disclosure-to-remediation gap in which reliable remediation still requires analyst involvement. Because production change windows often permit only one attempt before rollback, pass@5 should be interpreted as a research measure of recoverability rather than as a deployment-equivalent success rate. Removing the briefing substantially reduces performance: ReAct falls from 49.0% to 17.4% on MiniMax and from 56.0% to 13.0% on Qwen; Plan-and-Solve falls to 0% across every suite. Thus, when the agent must localise the vulnerable component without a per-host `threat.md`, zero-day performance is the more relevant deployment indicator. These results suggest that access to vulnerability context, such as CVE feeds, scanner output, and asset inventory, is at least as important as scaffold choice for operational accuracy. Plan-and-Solve's zero-day collapse further suggests that up-front planning is brittle when the initial vulnerability hypothesis is wrong. LATS performs worst at 3.2% pass@1, likely because tree-search rollouts divide the fixed wall-clock budget across branches and often terminate before a branch reaches verification.

The per-suite breakdown traces a security gradient. `ccdc` is most tractable (80–90% pass@1) because its scenarios reduce to documented hardening primitives (package upgrades, service disable, account policy edits) for which the model has strong priors; this is also the suite an attacker is least likely to find unpatched on a mature fleet, so high accuracy here moves the field forward least. The `ubuntu` subset of `meta3` is hardest among patchable suites (21–42%), dominated by configuration-edit scenarios in which file path, directive name, and restart sequence must all be correct at once; these are precisely the misconfigurations adversaries preferentially target, since operators reliably miss the same edits the agent misses. `meta4` (modern advisories across library, kernel, container, and AD surfaces) sits between (12–51%) and shows the largest @5 lift, indicating a nontrivial fraction of contemporary CVEs are recoverable from a second differently-seeded attempt and motivating ensembling for current threat material.

TABLE IV

HIVESTORM: REACT, ZERO-DAY ONLY. THE HIVESTORM SCORER ACCEPTS MULTIPLE INVOCATIONS DURING A SINGLE RUN; WE REPORT A SINGLE RUN PER SCENARIO. *SCORE* IS THE CUMULATIVE POINTS AWARDED AT RUN END, NORMALISED TO THE PER-SCENARIO MAXIMUM AND AVERAGED ACROSS THE 16 SCENARIOS.

Model	Score (%)	Resolved	Tot. tok	Wall
MiniMax-M2.7	52.3	1/16	17.2 M	7.43 h
Qwen3.5-9B	37.4	0/16	29.5 M	5.93 h

Table IV reports the Hivestorm suite separately because it is the hardest setting in SYSREPAIR: 16 multi-vulnerability hosts evaluated zero-day under a cumulative partial-credit rubric drawn from the published Hivestorm methodology, the closest analogue to an unaided incident-response queue. The headline scores (52.3% MiniMax, 37.4% Qwen) are partial-credit fractions rather than pass rates. Every other host ends with residual exploitable surface the aggregate conceals, the operational failure mode SYSREPAIR is built to expose: a pipeline reporting 52% mitigation may hand back zero clean hosts, with the audit trail recording activity without recording closure. The ceiling is also scaffold-independent in a deployment-relevant way: the better model resolves a single host across the suite and the smaller model resolves none, so growing parameter count does not move closure off zero when the agent must triage across multiple unannounced weaknesses on the same host. Cost compounds the result: 17.2M and 29.5M tokens against 7.43h and 5.93h per 16-host batch place per-host cost inside the exploitation window for high-severity CVEs, so unaided multi-vulnerability hardening is not yet a viable autonomous workflow even from the better-scoring scaffold. Current LLM agents add value here as a triage and partial-remediation layer rather than as a closure layer; this is where future work on the benchmark has the most room to move.

TABLE V

COST ON MINIMAX-M2.7, DAY-1@1. *TOKENS/SUCCESS* = TOTAL TOKENS DIVIDED BY SUCCESSFUL SCENARIOS. DOLLARS PER SUCCESS USE THE MINIMAX-M2.7 PRICING SNAPSHOT OF VI: $0.60/M INPUT, $2.40/M OUTPUT, $0.06/M CACHED INPUT; CACHED TOKENS ARE ESTIMATED AS *TOTAL − IN − OUT*.

Solver	Acc. (%)	Tot. tok	Wall	Tok/succ	$/succ
Reflexion	60.5	211M	4h10m	1.59M	$0.180
Basic	50.4	79M	2h12m	0.70M	$0.077
ReAct	49.0	126M	3h26m	1.05M	$0.120
Plan-Solve	28.6	2M	4h51m	27.8k	$0.028
LATS	3.2	5M	1h20m	0.63M	$0.533

B. Cost and efficiency

Table V reports total token consumption, wall-clock, and tokens-per-successful-scenario on the MiniMax-M2.7 Day-1@1 cells the information condition closest to operational deployment. The cost-versus-accuracy axis is stark: Plan-and-Solve resolves 28.6% of scenarios at ≈ 27,800 tokens per success, while ReAct resolves 49.0% at ≈ 1.05M tokens per success, a 38× token-cost ratio for a 1.7× accuracy ratio. Reflexion is the highest-accuracy solver at 60.5% pass@1 but the costliest per success at ≈ 1.59M tokens, because the reflection cycles add transcript volume without commensurate extra successes on scenarios that ReAct also resolves. LATS exhausts a similar wall-clock to Plan-and-Solve but converts almost none of it into oracle passes, yielding the worst tokens-per-success of any non-trivial solver. Figure 2 makes the trade-off visual: the Pareto frontier on this evaluation is occupied by Plan-and-Solve (cost-minimal), ReAct (balanced), and Reflexion (accuracy-maximal); LATS and the zero-day Plan-and-Solve cell are strictly dominated.

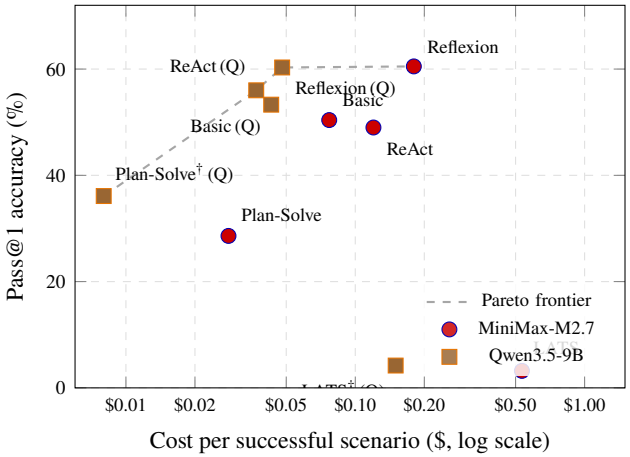

Fig. 2. Accuracy versus cost on Day-1@1 cells. Cost is dollars per successful scenario under the pricing snapshot of §VI. Square markers = Qwen3.5-9B; circle markers = MiniMax-M2.7. The Pareto frontier (dashed) now consists of four points: Qwen/Plan-Solve (cost-minimal), Qwen/ReAct, Qwen/Reflexion, and MiniMax/Reflexion (accuracy-maximal). Qwen/Plan-Solve dominates MiniMax/Plan-Solve on both axes; Qwen/Reflexion dominates MiniMax/Reflexion on cost at near-identical accuracy; MiniMax/Basic and MiniMax/ReAct are fully dominated by Qwen/ReAct. LATS is dominated for both models. †Qwen/Plan-Solve and Qwen/LATS costs are estimated from the MiniMax per-attempt token-usage ratios (Plan-Solve/ReAct = 0.136, LATS/ReAct = 0.290); accuracy values for these two cells are also estimated (see Table III).

C. Verifiability: NEUROPLAN on `ccdc`

NEUROPLAN is evaluated only on the 50-scenario `ccdc` subset and compared against the same subset on MiniMax-M2.7 and Qwen3.5-9B running ReAct, Day-1@5. Table VI reports four operating measurements that the LLM baselines do not produce: the rate at which the auto-generated PDDL parses and validates against the reference grammar (*validation rate*); the rate at which Fast Downward returns a plan within its 120 s budget (*plan rate*); the rate at which the lowered plan executes to an oracle pass on the live host (*exec rate*); and the mean number of operators in the emitted plan ($|\pi|$).

The PDDL pipeline succeeds end-to-end on 76% of CCDC scenarios, decomposed into 4% grammar misses, 8% lost at planning, and 12% lost at execution. The first two stages are *refusals*: NEUROPLAN halts with a structured outcome before any change reaches the host, so the operator escalates rather than reconciles partial state, the safe failure mode in change-controlled environments where a half-applied remediation leaves the host in a configuration downstream policy, monitoring, and IR tooling cannot trust. Only the 12% execution bucket modifies the

TABLE VI

NEUROPLAN ON ccdc (50 SCENARIOS) VERSUS REACT AT DAY-1 PASS@5 ON THE SAME SUBSET. *VAL.*, *PLAN*, *EXEC*: FRACTIONS OF SCENARIOS IN WHICH THE PDDL VALIDATES, A PLAN IS RETURNED, AND THE EXECUTED PLAN PASSES THE ORACLE. $|\pi|$ IS THE MEAN PLAN LENGTH FOR NEUROPLAN; $|\mathbf{c}|$ IS MEAN SHELL-COMMAND COUNT FOR THE REACT COMPARATORS. *TOK/SUCC* IS THE TOTAL TOKENS SUMMED ACROSS ATTEMPTS DIVIDED BY SUCCESSFUL SCENARIOS.

Solver	Val.	Plan	Exec	$\lvert\pi\rvert/\lvert\mathbf{c}\rvert$	Tok/succ
NEUROPLAN	0.96	0.88	0.76	41.4	442K
ReAct (MiniMax-M2.7) D1@5	–	–	0.92	193	2.12M
ReAct (Qwen3.5-9B) D1@5	–	–	0.89	221	2.53M

host, and the dual-objective oracle of IV-C catches it before misclassification as a closure. The inspectable artifact in either outcome is the lowered sequential plan and the typed PDDL it derives from: every action satisfies its precondition against the introspected state model by construction, Fast Downward refuses precondition-violating orderings without reaching the harness, and the plan exists *before* the host is touched. This pre-execution audit trail supports the change-management regimes under which the LLM baselines cannot operate, since their inspectable artifact is the during-execution transcript, and at pass@5 compounds to five trajectories the operator must reconcile after the fact.

Headline accuracy sits thirteen to sixteen points below ReAct at pass@5 (0.92 on MiniMax-M2.7, 0.89 on Qwen3.5-9B), but the framing is load-bearing: the LLM cells convert sampling stochasticity into accuracy across five resampled attempts, while NEUROPLAN is deterministic given a fixed domain and gains nothing from resampling, so the gap is a sampling-budget asymmetry rather than a reasoning-quality one. At that price, every NEUROPLAN closure arrives as a pre-execution plan and every non-closure as a single auditable transcript rather than up to five resampled trajectories. The compute differential reframes the trade further: a successful NEUROPLAN run executes 11.4 planned operators against ReAct's 193 and 221 cumulative shell invocations under the pass@5 budget, and at roughly 142K tokens per success it runs fifteen to eighteen times cheaper than ReAct on either model because the LLM is consulted at PDDL generation rather than per shell command, with most of that cost amortised offline. The accuracy gap is therefore the present price of verifiability, not its refutation, and the EW-loop and domain-coverage work in Appendix A-H together with the layered LLM-plus-NEUROPLAN posture flagged in VIII target closing it directly.

D. Failure-mode analysis

Across the failed runs in Table III we identify four modes consistent with the regime of II-A: (i) *ordering errors*, in which every individual command is well-formed but the sequence violates a dependency (firewall activated before allow rule installed; library upgraded before dependent app pinned) the hazard II-C identifies; (ii) *wrong-action errors*, in which the ordering is sound but the model chose an action that does not close the vulnerability (e.g. editing an inactive config path, restarting the wrong unit); (iii) *availability regressions*, in which the security oracle passes but a previously-operational service no longer responds; and (iv) *budget exhaustion*, in which the wall-clock cap of IV-D terminates the run before a coherent fix is emitted.

VIII. DISCUSSION

a) A three-axis decision rule for defenders: The results do not reduce to a single solver ranking. Reflexion provides the highest pass@1 accuracy, ReAct provides the strongest pass@5 accuracy, Plan-and-Solve provides the lowest token cost per success, and NEUROPLAN provides pre-execution plan inspectability within the evaluated ccdc subset. These are distinct deployment properties. A regulated environment that requires a reviewable change request before execution may prefer NEUROPLAN when the scenario lifts cleanly into its PDDL domain. A compute-constrained operator may prefer Plan-and-Solve despite its lower pass rate. An operator prioritising single-shot accuracy may prefer Reflexion. SYSREPAIR makes these trade-offs explicit rather than treating accuracy alone as the evaluation target.

b) Report-informed versus zero-day performance: The Day-1 to Zero-Day drop indicates that vulnerability context is a central constraint on autonomous remediation. In the report-informed condition, agents often produce a plausible fix once the vulnerable component is named. In the zero-day condition, they must also identify the relevant vulnerable surface within the same budget, and performance falls sharply. This suggests that operational deployments should evaluate not only the reasoning scaffold, but also the quality of the surrounding vulnerability-intelligence inputs, including scanner findings, CVE metadata, asset inventory, and configuration state.

c) Symbolic structure and its scope: Within the evaluated ccdc subset, NEUROPLAN demonstrates a different operating point from the reactive LLM solvers. It does not outperform ReAct on raw pass@5 accuracy, but it emits a pre-execution plan whose actions satisfy a typed precondition model before they reach the host. The guarantee is limited: it depends on valid PDDL generation, planner termination, and available shell lowerings, and it does not by itself prove production-level service correctness. The post-state must still pass the same security-and-availability oracle as every other solver. The ccdc result should therefore be read as evidence that this verifiability envelope is attainable on a defined subset, not as evidence that classical planning subsumes LLM agents across the full benchmark.

d) LLM and NEUROPLAN operating regimes: The two solver families fail in different regimes. LLM solvers are more flexible when remediation requires vendor-specific configuration edits, application-layer compensating controls, or kernel-version-specific patch sequences that are outside the current PDDL domain. NEUROPLAN is better suited to dependency-ordered repair sequences where typed preconditions can rule out unsafe orderings before execution. A practical deployment would likely combine both approaches: use NEUROPLAN when the scenario maps cleanly to the planning domain and pre-execution review is required, and use a ReAct- or Reflexion-

style solver when the remediation requires open-ended system investigation.

IX. Limitations and Threats to Validity

a) Single-host scope: SysRepair evaluates remediation on a single target host, consistent with the threat model of II-B. Production fleets introduce cross-host hazards (quorum-preserving rolling restarts, configuration-management drift, load-balanced replicas) we do not measure: a remediation safe on an isolated host can still violate availability across a cluster. Single-host scope is a lower bound on operational difficulty, and the ordering hazards of II-C are an intra-host subset of a larger problem.

b) Container and VM proxies: Every scenario builds from a pinned image rather than a live production stack. Containers and VMs reproduce the vulnerable software and its declared services but abstract away organic traffic, production monitoring, hardware-specific behaviour, and network complexity, so the availability oracle observes a quiescent rather than a loaded service. We treat the images as a faithful proxy for the security surface and an optimistic proxy for availability.

c) Residual nondeterminism: LLM sampling stochasticity is mitigated by five epochs per cell and frozen sampling parameters but not removed; pass@1 cells inherit it, and the cost figures of VII-B are means over a distribution. A small fraction of scenarios sit near the pass/fail boundary where the epoch outcome is effectively a coin flip.

d) Model drift: MiniMax-M2.7 is served through a vendor API whose weights and pricing can change without notice; the dollar figures of Table V are fixed to the snapshot of VI. The locally-served Qwen3.5 cells are pinned, reproducible from the released harness, and constitute the durable comparison.

e) NeuroPlan domain coverage: NeuroPlan is evaluated only on the 50-scenario `ccdc` subset, and its guarantee is conditional on a scenario lifting cleanly into a typed PDDL schema. Vendor-specific configuration directives, application-layer compensating controls, and kernel-version-specific patch sequences fall outside its domain envelope; the verifiability axis we claim holds on suites that lift cleanly, not across the full 313-scenario inventory.

f) Scenario-selection bias: The six suites inherit the composition of their public source corpora: a skew toward Linux server software, network-facing services, and vulnerability classes well represented in competition and scanner material. The `meta4` suite is authored by us, which risks encoding our own assumptions about tractable remediation; absolute pass rates should not be read as fleet-wide remediation rates.

g) Regression-oracle depth: The regression phase of IV-C is a preservation check rather than a full availability test suite. Each scenario pairs the security PoC with one or more post-remediation invariant checks (a daemon listening on a representative endpoint, a management path, a normal-user login, a compensating-control invariant), so a remediation can pass while leaving untested behaviour degraded such as latency, corrupted secondary state, or correctness only on the probed endpoint. Dual-objective scores therefore gate on remediation plus preservation of benchmark-specified invariants, not on full production-level semantic equivalence.

X. Ethics and Responsible Disclosure

SysRepair is built with the dual-use nature of autonomous security agents in mind. Every scenario instantiates an already-public vulnerability with an assigned identifier from the corpora of IV-A; we disclose no previously unknown vulnerabilities and weaponise no novel attack technique, and our contribution lies on the remediation side. Each canonical exploit drives only the security phase of the scoring oracle (IV-C) against a disposable, network-isolated target tightly coupled to the pinned scenario images, and is not packaged as a standalone retargetable offensive tool. An agent able to reconfigure, patch, and restart services could in principle be turned to harmful ends (log suppression, persistence, intrusion concealment), but the solvers we evaluate are standard published reasoning scaffolds and the shell action space is the ordinary administrative surface any root operator already possesses; SysRepair does not materially increase that risk, and its scoring regime is asymmetric in the defender's favour, rewarding only outcomes that close a disclosed vulnerability *and* preserve service availability. The full benchmark will be released under an open license on publication.

XI. Conclusion

SysRepair recasts the evaluation of autonomous defence as an explicit trade-off rather than a single accuracy ranking. Across 313 dual-objective scenarios scored under one protocol, no solver dominates on every axis: Reflexion attains the highest pass@1 (60.5%), ReAct the highest pass@5 (65.4%), Plan-and-Solve resolves 28.6% of scenarios at roughly 38× fewer tokens per success than ReAct, and, on the evaluated `ccdc` subset, NeuroPlan adds a verifiability axis by emitting a precondition-consistent plan that an operator can inspect before execution. Cost, accuracy, and verifiability are genuinely distinct properties, and the right solver depends on which a deployment requires. We release SysRepair so this choice can be made on measured evidence.

Acknowledgments

Acknowledgments are omitted to preserve anonymity for review and will be added in the camera-ready version.

References

[1] O. Setayeshfar, J. J. Rhee, C. H. Kim, and K. H. Lee, "Find My Sloths: Automated Comparative Analysis of How Real Enterprise Computers Keep Up with the Software Update Races," in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2021)*, vol. 12756 of *Lecture Notes in Computer Science*, pp. 215–236, Springer, 2021.

[2] A. Jenkins, L. Liu, M. K. Wolters, and K. Vaniea, "Not as Easy as Just Update: Survey of System Administrators and Patching Behaviours," in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*, (Honolulu, HI, USA), pp. 1–17, Association for Computing Machinery, 2024.

[3] N. Dissanayake, A. Jayatilaka, M. Zahedi, and M. A. Babar, "An Empirical Study of Automation in Software Security Patch Management," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*, pp. 7:1–7:13, Association for Computing Machinery, 2022.

[4] S. Beattie, S. Arnold, C. Cowan, P. Wagle, C. Wright, and A. Shostack, "Timing the Application of Security Patches for Optimal Uptime," in *Proceedings of the 16th USENIX Conference on System Administration (LISA '02)*, (Philadelphia, PA, USA), pp. 233–242, USENIX Association, 2002.

[5] Y. Zhu, A. Kellermann, D. Bowman, *et al.*, "CVE-bench: A benchmark for AI agents' ability to exploit real-world web application vulnerabilities," in *Proc. International Conference on Machine Learning (ICML)*, 2025. Spotlight. arXiv:2503.17332.

[6] Z. Liu, J. Shi, and J. F. Buford, "CyberBench: A multi-task benchmark for evaluating large language models in cybersecurity," in *AAAI-24 Workshop on Artificial Intelligence for Cyber Security (AICS)*, 2024.

[7] P. Jing *et al.*, "SecBench: A comprehensive multi-dimensional benchmarking dataset for LLMs in cybersecurity," *arXiv preprint arXiv:2412.20787*, 2024.

[8] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can language models resolve real-world GitHub issues?," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. Oral. arXiv:2310.06770.

[9] X. Deng, J. Da, E. Pan, *et al.*, "SWE-bench pro: Can AI agents solve long-horizon software engineering tasks?," *arXiv preprint arXiv:2509.16941*, 2025.

[10] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.15793.

[11] M. S. Boddy, J. Gohde, T. Haigh, and S. A. Harp, "Course of action generation for cyber security using classical planning," in *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 12–21, 2005.

[12] Y. Liu, Y. Guo, R. Ranjan, and D. Chen, "Optimization of mitigation deployment using deep reinforcement learning over an enhanced ATT&CK," *Computing*, vol. 106, pp. 4015–4038, 2024.

[13] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

[14] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.11366.

[15] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim, "Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models," in *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. arXiv:2305.04091.

[16] A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang, "Language agent tree search unifies reasoning acting and planning in language models," in *Proc. International Conference on Machine Learning (ICML)*, 2024. arXiv:2310.04406.

[17] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, "Agentless: Demystifying LLM-based software engineering agents," in *Proc. ACM International Conference on the Foundations of Software Engineering (FSE)*, 2025. ACM SIGSOFT Distinguished Paper Award. arXiv:2407.01489.

[18] DARPA, "AI cyber challenge (AIxCC) final competition results." https://www.darpa.mil/news/2025/aixcc-results, 2025. Reported at DEF CON 33, August 2025.

[19] T. Avgerinos, D. Brumley, J. Davis, R. Goulden, T. Nighswander, A. Rebert, and N. Williamson, "The Mayhem cyber reasoning system," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 52–60, 2018. Mayhem won the DARPA Cyber Grand Challenge final, August 2016.

[20] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *Proc. USENIX Security Symposium*, 2024. Distinguished Artifact Award.

[21] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. A. Marriott, "CybORG: A gym for the development of autonomous cyber agents," *arXiv preprint arXiv:2108.09118*, 2021.

[22] M. Kiely, M. Ahiskali, E. Borde, B. Bowman, D. Bowman, D. Van Bruggen, K. Cowan, P. Dasgupta, E. Devendorf, B. Edwards, A. Fitts, S. Fugate, R. Gabrys, W. Gould, H. H. Huang, J. Jacobs, R. Kerr, I. J. King, L. Li, L. Martinez, C. Moir, C. Murphy, O. Naish, C. Owens, M. Purchase, A. Ridley, A. Taylor, S. Farmer, W. J. Valentine, and Y. Zhang, "Exploring the efficacy of multi-agent reinforcement learning for autonomous cyber defence: a cage challenge 4 perspective," in *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25, AAAI Press, 2025.

[23] N. Merrill *et al.*, "Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces," *arXiv preprint arXiv:2601.11868*, 2025.

[24] Mondoo, "Introducing agentic vulnerability patching using Ansible." https://mondoo.com/blog/introducing-agentic-vulnerability-patching-using-ansible, 2025.

[25] Qualys, "Meet agent val: Closing the validation gap in exposure management at machine speed with agentic AI." Qualys Blog, March 2026, https://blog.qualys.com/, 2026.

[26] P. E. Kaloroumakis and M. J. Smith, "Toward a knowledge graph of cybersecurity countermeasures," tech. rep., The MITRE Corporation, 2021.

[27] The MITRE Corporation, "Update software, mitigation M1051 – enterprise." MITRE ATT&CK®, 2024.

[28] National Institute of Standards and Technology, "Risk management framework for information systems and organizations: A system life cycle approach for security and privacy," Tech. Rep. NIST SP 800-37 Rev. 2, U.S. Department of Commerce, 2018.

[29] The MITRE Corporation, "Exploit public-facing application, technique T1190 – enterprise." MITRE ATT&CK®, 2024.

[30] M. Souppaya and K. Scarfone, "Guide to enterprise patch management planning: Preventive maintenance for technology," Tech. Rep. NIST SP 800-40 Rev. 4, National Institute of Standards and Technology, 2022.

[31] The MITRE Corporation, "Update software, mitigation M0951 – ics." MITRE ATT&CK®, 2024.

[32] B. Liu, Y. Zhao, G. Xu, and H. Wang, "LLM agents for automated web vulnerability reproduction: Are we there yet?," *arXiv preprint arXiv:2510.14700*, 2025.

[33] J. Loevenich, E. Adler, T. Huerten, and R. R. F. Lopes, "Design and evaluation of an autonomous cyber defence agent using DRL and an augmented LLM," *Computer Networks*, 2025.

[34] T. Xu, Z. Wen, X. Zhao, J. Wang, Y. Li, and C. Liu, "L2M-AID: Autonomous cyber-physical defense by fusing semantic reasoning of large language models with multi-agent reinforcement learning," *arXiv preprint arXiv:2510.07363*, 2025.

[35] N. Potteiger, A. Samaddar, H. Bergstrom, and X. Koutsoukos, "Designing robust cyber-defense agents with evolving behavior trees," in *Proc. IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2024. arXiv:2410.16383.

[36] M. T. Alam, D. Bhusal, L. Nguyen, and N. Rastogi, "CTIBench: A benchmark for evaluating LLMs in cyber threat intelligence," in *Proc. Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. Spotlight. arXiv:2406.07599.

[37] National CCDC, "National collegiate cyber defense competition (CCDC)." https://www.nationalccdc.org/, 2024. Accessed: 2024-05-15.

[38] Rapid7, "Metasploitable 2." https://docs.rapid7.com/metasploit/metasploitable-2/, 2012. Exploitation training environment.

[39] VulnHub, "VulnHub: Material for security research." https://www.vulnhub.com/, 2024. Community-driven vulnerable virtual machines.

[40] Rapid7, "Metasploitable 3." https://github.com/rapid7/metasploitable3, 2016. Open-source vulnerable virtual machine build framework.

[41] Hivestorm, "Hivestorm cyber defense competition." https://hivestorm.org/, 2024. Collegiate cyber defense challenge.

[42] UK AI Safety Institute, "Inspect: An open-source framework for large language model evaluations." https://inspect.aisi.org.uk/, 2024.

[43] osquery Authors, "osquery: SQL powered operating system instrumentation, monitoring, and analytics." https://github.com/osquery/osquery, 2014–2026. Linux Foundation project. Accessed 18 May 2026.

[44] AI Planning Community, "pddl: an unquestionable PDDL 3.1 parser." https://github.com/AI-Planning/pddl, 2020–2026. Accessed 18 May 2026.

[45] M. Helmert, "The fast downward planning system," *CoRR*, vol. abs/1109.6051, 2011.

[46] S. Mahdavi, R. Aoki, K. Tang, and Y. Cao, "Leveraging environment interaction for automated pddl translation and planning with large language models," 2024.

[47] Vast.ai, "Rent L40 GPUs on Vast.ai for $0.40/hr." https://vast.ai/pricing/gpu/L40, 2026. Accessed 18 May 2026.

Appendix A
NeuroPlan Implementation in Detail

This appendix expands the six-stage pipeline of Section V-B into a reproduction-grade specification: lifting the live host into a typed state model (A-A), mining and merging the domain from `man(1)` pages (A-B), tightening preconditions (A-C), the type and predicate vocabulary (A-D), per-scenario problem lifting (A-E), the planner configuration (A-F), lowering to shell (A-G), the exploration-walk refinement loop (A-H), and end-to-end orchestration and validation (A-I, A-J).

A. State Introspection and Anchor-and-Propagate Scoping

Introspection runs an osquery 3.1.1 extension over Thrift, exposing the standard SQL tables (`packages`, `services`, `listening_ports`, `users`, `groups`, `processes`, `file`, `interface_addresses`); a shell fallback (`osqueryi --json`) covers container builds without the Thrift socket. Each typed record is converted into a ground predicate, one per row.

A raw snapshot is too large to ground efficiently ($\sim 10^3$ packages, hundreds of scoped files, dozens of unused accounts), so NeuroPlan applies an *Anchor-and-Propagate* reduction before emitting state. (i) *Anchors* are the user-facing entities—every listening socket, every active `service`, every `human_user` (non-system UID), and root. (ii) A directed *dependency graph* is built over the typed entity set from ownership (`file_owned_by`, `process_owned_by`), control (`service_depends_on_package`, `configures`), and membership (`member_of`) edges. (iii) A BFS sweep marks every entity *reachable* from the anchors. (iv) Unmarked entities are *pruned*, and the pre/post entity counts are recorded in the state metadata.

The output is a lifted state with three fields: a map from each type to its surviving identifiers, the ground predicates, and the typed edges retained for downstream enrichment. A *legacy* mode using static per-type caps is kept only as the ablation baseline in Table III.

B. Domain Generation: Map–Reduce Mining and Merging

The domain is built in two LLM stages; each must parse against the `pddl` reference grammar [44] before admission.

Phase 1 (map). A man-page parser reads `man(1)` entries for six utility groups: package (`apt-get`, `dpkg`), service (`systemctl`, `service`), file (`chmod`, `chown`, `touch`, `rm`), network (`iptables`, `ip`), privilege (`sudo`, `su`), and user (`useradd`, `usermod`, `passwd`, `groupadd`). Extraction uses the `langextract` library on the configured LLM (Section VI); for MiniMax models, `<think>` reasoning blocks are stripped before JSON parsing. Each record is a typed action schema with a parameter list, a positive precondition set, an effect set, and a shell-command template harvested from the man-page SYNOPSIS.

Phase 2 (reduce). One worker per group is given the canonical type set and predicate signatures and constrained to draw only from them; `(assert...)`, `(equal...)`, quantifiers, and implications are blocked at the prompt and re-checked at parse time. Workers may seed from the Phase 1 schemas. The merger then (1) repairs syntax via the deterministic sanitiser of A-J; (2) unifies types against the canonical hierarchy, coercing hallucinated names; (3) deduplicates predicates by signature, dropping any with undeclared argument types; (4) merges colliding operators by *intersecting* preconditions—the conservative direction, which never loosens an operator—and *unioning* effects; (5) injects hand-engineered *canonical actions* (e.g. `edit_config_setting`, `reload_sshd_no_systemd`) for primitives no single man page surfaces cleanly; and (6) validates the merge end-to-end, regenerating unparseable operators up to a bound before dropping and logging them. The result is a single typed, deduplicated, grammar-clean domain.

C. Enrichment and Precondition Synthesis

Two LLM-free passes tighten the merged domain. *Environment enrichment* probes the scenario container for assumed capabilities (systemd, `iptables`, `apt`, `snap`, an active firewall, MAC enforcement), maps each operator's leading command token to a capability, and conjoins the matching 0-arity predicate (e.g. `systemd_init_present`) into its `:precondition`; the pre-enrichment domain is preserved for auditability. This is what stops the planner from emitting a `systemctl` plan against a `sysvinit` container—the operator is refused symbolically rather than failing at runtime.

Rule-based synthesis encodes a closed table for canonical actions the man-page mining under-specifies: `useradd`/`groupadd`/`touch` require the target *not* to exist; `passwd`, `chage`, `usermod` require the user to exist and not be `user_critical`; `su` requires the target to exist and satisfy `can_login`; `rm`/`chmod` require `file_exists`. The table is matched against command token and parameter signature in one static pass: man pages document the *flags*, not the *state* the operator assumes.

D. Type Hierarchy and Predicate Vocabulary

The fixed hierarchy is `object` $\supset$ {`filesystem_object` $\supset$ {`file`, `directory` $\supset$ `configuration_file`}, `service`, `process`, `package`, `repository`, `user` $\supset$ {`system_user`, `human_user`}, `group`, `port`, `interface`, `firewall_rule`}.

The predicate vocabulary splits into: *static relations* set at introspection and never rewritten (`depends_on`, `configures`, `file_owned_by`, `member_of`); *dynamic state* targeted by operator effects (`package_installed`, `package_outdated`, `vulnerable`, `service_running`/`_enabled`/`_failed`, `config_applied`, `file_exists`/`_readable`/`_writable`/`_permissions`, `user_exists`, `user_critical`, `can_login`, `group_exists`, `setting_value_is`); and *environment capabilities* (0-arity, set by enrichment, e.g. `apt_available`). The split realises the verifiability contract: static relations are evidence, dynamic state is the surface the plan traverses, capabilities close the domain over what the host supports.

E. Problem Generation

Per scenario, NeuroPlan synthesises a problem from the lifted state and the scenario brief. The prompt gets a compact domain summary (types, predicate and operator signatures with truncated bodies), the vulnerability description, and the current snapshot from a fixed osquery probe set; the domain is fixed, so the LLM emits only `:objects` (goal-referenced constants restricted to the type set), `:init` (ground facts for the introspected state plus goal prerequisites), and `:goal`. The goal conjoins (i) the *negated exploit predicate* (e.g. `(setting_value_is PermitRootLogin no)`) and (ii) an *availability predicate* for every service operational in the pre-state (e.g. `(service_running sshd)`); the second conjunct forces the planner to reject obvious-but-destructive fixes such as stopping the service. The problem is parsed against the unified domain; one regeneration is allowed on a parse error, after which the run halts with the structured outcome `PROBLEM_GENERATION_FAILED`.

F. Classical Planner Configuration

Planning uses Fast Downward [45] as a sub-process configured for `eager_greedy([ff()])`: eager greedy best-first search under the FF relaxed-plan heuristic, a 120 s default budget bounded above by the harness budget of IV-D. The lowest-cost plan is parsed into grounded operator invocations; no plan within budget yields the outcome `PLANNER_FOUND_NO_PLAN`. Heuristic inadmissibility is acceptable because every plan is validated by execution against the oracle of IV-C: we seek a satisfiable plan, not an optimal one.

G. Plan-to-Shell Lowering

Each grounded operator is lowered through a three-tier dispatcher. *Tier 0* reuses the verbatim command template harvested in Phase 1 when name and arity match. *Tier 1* is a static table of ∼20 canonical lowerings (e.g. `install_package` ↦ `apt-get install -y {0}`; `restart_service` ↦ `(service {0} restart 2>/dev/null)||systemctl restart {0}`; `edit_config_setting` ↦ a parameterised `sed -i`), with type-aware normalisers that resolve `sshd_config` → `/etc/ssh/sshd_config` and case-fold setting keys, stripping PDDL separators before substitution. *Tier 2* is a one-shot LLM concretiser fallback given the operator's PDDL, bound parameters, brief, and an environment probe; its output is sanitised (`<think>` blocks, code fences, and a reflexive `sudo` prefix removed—the harness already runs as `root`) and rejected if it equals the literal operator name. Successful lowerings are cached against the signature.

H. Exploration-Walk Refinement of the Domain

A typed precondition model is necessary but insufficient: an operator can satisfy every symbolic precondition yet regress at runtime via init-script timing, contention, or a reload dropping in-flight connections. NeuroPlan closes this with an *exploration-walk* (EW) loop [46] run once per domain, *offline*, before any scenario.

A PDDL simulator keeps a closed-world fact set and per-type object pools, and at each step applies a three-tier filter: (1) drop operators with absent precondition predicates; (2) ground surviving operators against the type pools; (3) verify the full precondition conjunction. A fully applicable grounding is sampled uniformly, lowered (A-G), executed live, and the state updated by its add/delete effects. The predicted post-state is compared to a follow-up osquery probe; an operator that *fails* despite satisfied preconditions (non-zero exit, missing effect) is logged as a *discrepancy* with a structured failure mode. The refiner runs N walks of depth $T_{\max}$ (auto-scaled to domain size or configured); the EW score is the fraction of PDDL-legal groundings that succeeded in the container. Persistently failing operators trigger a per-action repair LLM call (≤100/iteration); the repaired domain is re-validated and the loop iterates until the score clears a threshold (default 0.9) or the budget is spent, rolling back to the best domain seen on catastrophic regression. The refined domain is frozen for scenario time with no further adaptation.

I. End-to-End Orchestration

The pipeline runs three phases in disposable containers: *A* (fresh base)—introspection and man-page mining, yielding the lifted state and seed schemas; *B* (fresh base)—parallel utility-group workers, merge, canonical-action injection, environment enrichment, and rule synthesis, yielding the merged domain; *C* (scenario image)—the EW loop, yielding the refined domain.

The neurosymbolic solver is registered as an Inspect AI [42] solver, so its per-scenario call signature matches the four LLM baselines of V-A. It loads the refined domain, runs the fixed introspection probe set, generates the problem (A-E), invokes the planner (A-F), reorders the plan once to push configuration edits before service restarts (a dependency STRIPS does not capture), and executes up to 20 lowered actions through the same shell channel as the baselines. Verification is delegated to the scenario's verifier, terminating with one of `NO_DOMAIN`, `PROBLEM_GENERATION_FAILED`, `PLANNER_FOUND_NO_PLAN`, `PLAN_DID_NOT_REMEDIATE`, or `REMEDIATED`. The task wrapper reuses the baselines' sample loader, scorer, and rate limiter, so the harness, budget, and oracle of IV-D/IV-C apply unchanged and the full evaluation reduces to a single benchmark command against a chosen model, holding the comparison to the LLM solvers under one protocol.

J. PDDL Sanitisation and Validation

LLM-emitted PDDL fails parsing in known ways, so a deterministic sanitiser applies rewrites before any text reaches the parser: reserved-keyword substitution (`((exists...)` ↦ `(file_exists...)`, `(start...)` ↦ `(is_started...)`); identifier normalisation (paths such as `/etc/passwd` → `etc_passwd`, special chars → underscores, case folded on type names and preserved on user constants); type repair mapping hallucinated

names (`string`, `boolean`, CamelCase variants) to the type set of A-D; and predicate repair (paren-balanced extraction, variable-scope validation, malformed-effect removal). Final validation against the `pddl` library is deliberately defensive: an empty types or predicates list is treated as a silent parse failure and propagated as a structured failure, letting the merger of A-B recover from per-worker noise without inheriting it into the planning surface.

K. Summary of Verifiability Guarantees

The contract of Section V-B reduces to four enforced obligations. (1) Every emitted operator satisfies a typed precondition conjunction tightened by man-page mining (A-B), capability enrichment, and rule synthesis (A-C); precondition-violating orderings are refused by Fast Downward (A-F) and never reach the harness. (2) Domain and problem are statically validated against the `pddl` grammar (A-J) before planning, with structured outcomes rather than silent skips. (3) The operator library is empirically reconciled with the live host by the EW loop (A-H) before evaluation. (4) The post-execution state is scored by the same oracle (IV-C) as every LLM baseline. For each NEUROPLAN run the released artifact retains the lifted state, the refined domain, the generated problem, the Fast Downward plan, the lowered command stream, and the oracle verdict, so that every link of the chain is auditable post hoc.