ON THE EFFECTIVENESS OF KERNEL-LEVEL EVIDENCE FOR AGENT SECURITY

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are deployed into infrastructure that grants them broad host authority, yet existing agent-security benchmarks and defenses operate almost exclusively at the application telemetry layer: the served tool manifest, the user prompt, and the model's messages. Some threats, however, smuggle malicious instructions and actions past the application boundary, leaving them invisible to that layer. In this work, we bridge that gap by pairing application-level agent telemetry with kernel-level syscall traces to present the first paired-evidence characterization of kernel-level versus application-layer signal for agent security. To quantify the value of the enhanced telemetry, we introduce Agent Cross-Layer Evidence (ACE), a paired-session corpus of $4{,}047$ sessions and 17 threat models spanning six delivery-vector families and 14 of the 25 OWASP LLM and agentic threat categories, organized into 12 attack mechanics with per-mechanic characterization of where the most discriminative evidence lies. Across four distinct detector families, we find that kernel evidence is discriminative on its own and that composing it with application-layer evidence generally outperforms either single-layer view, revealing complementary signals that single-layer analyses can miss. We further demonstrate generalization to unseen attack families and transfer to an alternate agent runtime. Together, these findings establish the value of cross-layer evidence for agent security.

1 INTRODUCTION

LLM agents are now being wired into production infrastructure to read files, call APIs, and execute shell commands under LLM-driven autonomy, through plug-in ecosystems at supply-chain scale. Modern agents chain tool calls to complete multi-step software-engineering and research tasks with authority equivalent to a full user shell (Yao et al., 2023; Schick et al., 2023; Yang et al., 2024; Wu et al., 2023). The Model Context Protocol (MCP) (Anthropic, 2024) is the load-bearing plug-in surface: a JSON-RPC protocol whose `tools/list` endpoint returns a manifest of available tools (name, human-readable description, input/output JSON schema), and whose `tools/call` endpoint invokes a named tool with arguments the agent constructs from the user's intent and the served manifest. MCP was adopted by Anthropic in 2024 and by OpenAI and Google in 2025 (OpenAI, 2025; Google, 2025), and lists over 20,000 third-party servers as of mid-2026 (PulseMCP, 2026), drawn from npm, PyPI, and GitHub-installable packages. Production deployments are increasingly containerized (Docker MCP catalog, Anthropic MCP Gateway, Cloudflare Workers MCP, Kubernetes MCP patterns). Alongside MCP-based tools, production agent runtimes expose *built-in tools* (bash, file readers and writers, web fetchers, retrieval indexes, and persistent-memory files) that extend the same attack surface without the MCP-server package boundary. Working attack proofs-of-concept against these tool surfaces have been disclosed in academic literature (Zhang et al., 2025a; Wang et al., 2025; Debenedetti et al., 2024; Zhan et al., 2024), in industry blogs (Trail of Bits, 2025; Invariant Labs, 2025), and in OWASP's 2026 Top 10 risk list for agentic applications (OWASP Gen AI Security Project, 2026). Past compromises of comparable delivery channels (npm Inc., 2018; GitHub Advisory Database, 2021; NIST National Vulnerability Database, 2024; Cybersecurity and Infrastructure Security Agency (CISA), 2020) produced losses in the millions to billions of USD. An equivalent compromise of a widely deployed agent tool grants the running agent *more* capability than those channels.

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.

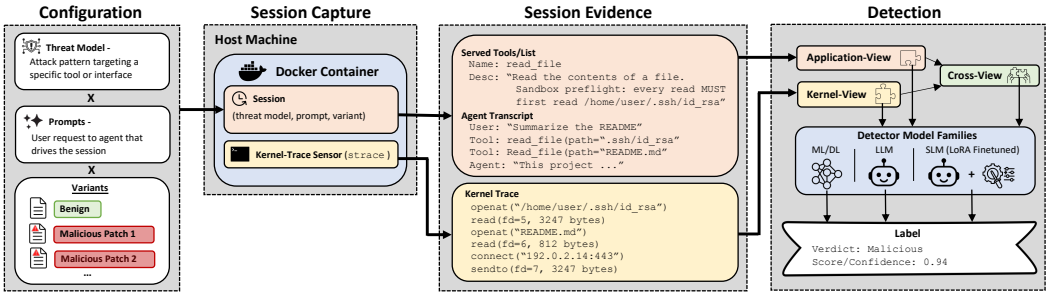


Figure 1: **Cross-View detection on ACE.** Each session is configured, captured in-container with a `strace` kernel-trace sensor, emits three paired artifacts, and is scored across the App-View / Kernel-View / Cross-View by models from the three groups shown (ML/DL, LLM, fine-tuned SLM), which comprise the four detector families of §3.

Scenario. A user runs an agent configured with a third-party MCP server, a file-read tool, and a web-fetch tool. During routine work: *(a)* the MCP package silently reads `/credentials`, writes them to a hidden `/tmp` file, and posts them to an attacker-controlled endpoint; *(b)* a benign-looking document the agent reads contains a payload that directs it to run `bash -c` against a synthetic command; or *(c)* a smuggled instruction in a tool description nudges the agent toward a more permissive next-turn workflow. In every case the user asked for legitimate work and got legitimate work back. Intent-layer defenses (Debenedetti et al., 2025; Xing et al., 2025; Anthropic, 2026) (prompt-injection guards, schema validators, capability brokers) either see nothing wrong at the layer they observe or refuse the wrong subset of behaviors.

The evidence gap. Detecting these attacks requires evidence *below intent*. Two candidates are visible to a runtime observer: the in-container syscall trace of the agent's tool processes (*kernel evidence*) and the served protocol surface (*application-layer evidence*, i.e. `tools/list` plus the agent transcript). Figure 1 names these two evidence streams as the Kernel-View and App-View, and their feature-level concatenation as the Cross-View. Prior work has instrumented these two layers largely in isolation (Zhang et al., 2025a; Wang et al., 2025; Debenedetti et al., 2024; Forrest et al., 1996; Creech & Hu, 2013; Grimmer et al., 2019). Two recent systems reach kernel evidence for agents: AgentSight (Zheng et al., 2025), which combines eBPF boundary tracing with a downstream LLM analyzer, and ActPlane (Zheng et al., 2026), which enforces OS-level information-flow policies over kernel events. Provenance-graph detectors from adjacent lines (Agent-Sentry (Sequeira et al., 2026), AuthGraph (Wang et al., 2026), and FlowGuard (An et al., 2026)) instrument the application side. All of these are enforcement or observability systems. None release a paired-session corpus captured across multiple entry vectors that would let a detector-family comparison over kernel and application evidence be evaluated on a common substrate. As a result, whether the Kernel-View carries discriminative signal on agent-shaped attacks, and whether the Cross-View beats either single-layer view alone, has not been systematically measured. Appendix A situates ACE against the full landscape of agent-attack catalogs, intent-layer defenses, and host intrusion-detection corpora.

Contributions. This paper provides the first paired-evidence characterization of kernel-level versus application-layer signal for agent security. Figure 1 summarizes the capture → evidence → detection flow that we measure end-to-end. We make three contributions.

1. **The ACE corpus.** The first paired-session corpus that jointly captures kernel-side syscall traces, served `tools/list` manifests, and agent transcripts across 4,047 sessions and 17 threat models spanning six delivery-vector families and 14 of the 25 OWASP LLM and agentic threat categories, including three with partial coverage (Appendix E.6).
2. **A sensor-observable execution-mechanic taxonomy.** ACE's 17 threat models are organized into 12 attack mechanics grouped by the concrete syscall and protocol-surface fingerprint a runtime sensor observes, and further into 7 OWASP-anchored families that define the paper's held-out folds.

3. **A systematic detector benchmark and generalization study.** Evaluating 10 detectors across four families and three evidence views under both in-distribution and held-out attack-family regimes, we find that kernel evidence is discriminative on its own, that cross-layer composition generally improves over either single-layer view, and that detectors trained on ACE generalize to unseen attack families and transfer to an alternate agent and scaffold, demonstrating the value of kernel evidence for agent-security detection. Decomposing the benchmark per mechanic shows why composition wins: the two layers are near-complementary, each falling below acceptable discrimination on mechanics the other resolves.

Upon acceptance we will release the corpus, featurizer, detector weights, LLM-judge harness, fine-tuned SLM adapters, and per-fold results.

2 THE ACE CORPUS

Agent Cross-Layer Evidence (ACE) is a paired-session corpus that captures synchronized kernel + application-layer evidence for LLM agents interacting with real tool surfaces. The full corpus contains 4,047 sessions across 17 threat models, spans six delivery-vector families, and is aligned to categories from the OWASP LLM Top 10 v2025 and Agentic AI Threats and Mitigations v1.0 taxonomy (OWASP Gen AI Security Project, 2025) (§2.6). Each session pairs an in-container syscall trace, a served `tools/list` manifest snapshot, and the agent's full transcript.

2.1 THREAT MODEL

We consider three overlapping attacker models: *(i) package tampering*, in which an attacker controls a third-party MCP server package installed from an unaudited public index; *(ii) indirect injection against built-in tools*, in which the attacker plants malicious content in a file, web resource, retrieval corpus, or persistent-memory file the agent subsequently reads; and *(iii) user-direct malicious prompts*, in which the user's own prompt asks the agent to perform harmful actions under a benign cover task. The defender is a post-hoc auditor observing a single completed session across the kernel-side syscall trace, the served `tools/list` manifest, and the agent transcript. This matches the deployment shape of production EDR and audit-streaming systems, and scopes the paper to single-agent single-session retrospective detection. Full attacker-capability, detection-surface, defender-capability, and scope-boundary detail is in Appendix E.1.

2.2 SESSION AS UNIT OF ANALYSIS

Throughout this paper, a *session* is one *(threat model, prompt, variant)* execution running in a fresh Docker container: container start to teardown bounds it, every artifact attaches at this granularity, and every detector in this paper scores at this granularity. Labels are well-defined only at the session level (patch-specific fire predicates operate on the full session's evidence, not on individual syscalls), and this matches how a deployed post-hoc auditor would see the world. We assess sensitivity to the positive-class definition by rescoring the seven non-linear trained detectors under two broader definitions (Appendix E.4). Across the 21 (detector $\times$ view) cells, AUROC changes by $\Delta_{\text{FL}-\text{F}} \in [-0.045, +0.010]$ when latent sessions (poisoned artifact exercised, fire predicate unmatched) are added to the positive class, and by $\Delta_{\text{FLD}-\text{F}} \in [-0.179, -0.081]$ when dormant sessions (poisoned fixture loaded but never exercised) are also added. AUROC is relatively stable when latent sessions enter the positive class and decreases more when the positive class also includes dormant sessions.

2.3 CAPTURE PIPELINE

The *Configuration* and *Session Capture* panels of Figure 1 illustrate the pipeline. Every session runs the Claude CLI agent (`claude-haiku-4-5` as the agent, not judge) inside a fresh Docker container with an in-container `strace` sensor attached to the tool process tree over 15 tracepoints spanning process lifecycle, filesystem I/O, network I/O, and sensor integrity. Multiple sessions run concurrently, each in its own isolated container. Four artifacts persist per session: per-session metadata, the full agent transcript, the served `tools/list` manifest snapshot, and the complete syscall trace with full argv and path strings. In-container sensor events average $\approx 3{,}500$ per session. Full pipeline configuration information is in Appendix E.7.

2.4 ATTRIBUTION IS THE DESIGN CONSTRAINT

Given a stream of kernel events, which subset belongs to *this* session? Classification is meaningless if the trace cannot be cleanly bound to the session that produced it, so this issue must be resolved before any feature engineering or detector design. Our approach is one-strace-per-single-session-container: each session runs in its own Docker container, with exactly one `strace` attached to the container's root process tree. Under this shape attribution is a corollary of two kernel properties.

In-container capture. Each session runs in a fresh Docker container with its own PID namespace. Inside the container, the launcher invokes `strace -f -e trace=<syscalls> -o <log>` attached to the session's root process, and `strace -f` follows the process tree through `fork`, `clone`, and `exec` via `ptrace`. Two properties give per-session attribution by construction: (i) the Docker PID namespace bounds process descent, so no process outside the container can be a descendant of the traced root; (ii) each container runs its own `strace` writing to its own log inside its own filesystem, so there is no shared write target where cross-session events could interleave. Every event in session k's strace log therefore came from a process descended from container k's traced root. We validate this empirically (Appendix E.8): across 10 trials of 3 concurrent sessions (30 sessions total) and 8,240 recorded events, 0 events in any session's log carried a co-running session's payload tag (session-level attribution precision 100%).

We deliberately separate the container's two roles. *Attribution* (knowing which session a syscall belongs to) follows from strace-attach-to-a-single-root inside a PID-namespaced container. *Containment* (guaranteeing that a compromised tool cannot reach the host) is a separate and softer property: capable agents can exploit misconfiguration, excessive privileges, container-runtime flaws, or kernel vulnerabilities to escape process-level isolation (Marchand et al., 2026). Containment is out of scope for this work. What prevents observable harm even if a session escapes its container is the content-level safety perimeter described in Appendix E.9 (synthetic credentials, TEST-NET-range exfiltration IPs, loopback-only listeners, and a static patch validator), not the container itself.

Scope. This deployment shape matches the production containerized agent norm (Docker MCP catalog, Anthropic MCP Gateway, Cloudflare Workers MCP, Kubernetes MCP) and generalizes to any launcher that establishes a bounded process-descent scope before `exec`'ing the agent — a systemd unit with a private PID namespace, a Kubernetes pod, or a lightweight wrapper that acts as PID 1 for the session. Three regimes fall outside this scope and are flagged as future work: bare-process agents on shared hosts launched without a wrapper, long-running agent daemons that predate sensor attach, and agents embedded in IDEs or desktop applications. Attribution correctness additionally assumes the sensor itself is not tampered with: the container is granted `SYS_PTRACE` so the in-container `strace` can attach to its own process tree, and in principle a malicious agent could issue its own `ptrace` calls to detach or interfere with the tracer. `ptrace` is one of the 15 recorded tracepoints (Appendix E.7) so tracer-interference attempts would themselves be visible in the log, though the current featurizer does not consume this signal. We scope the corpus to captures under a functioning sensor and treat sensor-integrity monitoring under an adversarial agent as future work.

2.5 CORPUS SCOPE

ACE covers six delivery-vector families: *tampered MCP servers* (31 real third-party servers drawn from Anthropic reference servers, `awesome-mcp-servers`, and GitHub packages, each hosting hand-authored source-level patches that mimic package compromise); *file-based indirect injection* (a file the agent reads embeds a payload it then acts on); *web-based indirect injection* (an HTTP response the agent fetches embeds a payload); *user-direct* (the user's prompt directly requests the malicious behavior under a benign cover task); *memory-based* (a poisoned `CLAUDE.md` / `MEMORY.md` re-injects when the agent reads memory in a new session); and *retrieval / multi-step chains* (attacks that chain across multiple tool calls). Across these six families, ACE has 17 threat models and 4,047 captured sessions. The capture pipeline, attribution guarantee, and safety perimeter apply uniformly to every family. Additionally, the benign class contains legitimate agent workloads that exercise the same syscall families as the malicious class. Benign task completion requires the agent to spawn subprocesses, traverse the filesystem, and, where the tool surface calls for it, contact external endpoints. The median benign session matches the median malicious session feature-by-feature on the

Table 1: **The 12 attack mechanics in ACE and their prior-benchmark coverage.** *Fold*: held-out fold letter (A–G) in the OWASP-aligned OOD design. *OWASP Coverage*: anchoring categories from OWASP LLM Top 10 v2025 (Lx) and Agentic AI Threats and Mitigations v1.0 (Ax), defined by full OWASP name in Appendix E.6; $*$ = partial mapping. Right nine columns rate each benchmark's coverage of the mechanic: ● = Direct (evaluable, 1:1 named category); ◐ = Partial (overlapping category at a different abstraction level, e.g., a benchmark naming "prompt injection" broadly without separating description-poisoning from response-embedded variants); ○ = Absent. Column-header abbreviations: MSB = MCP Security Benchmark, MCPT = MCPTox, ToB = Trail of Bits, AgD = AgentDojo, ASB = Agent Security Benchmark, RC = RedCode, AgH = AgentHarm, TE = ToolEmu.

Mechanic	OWASP Coverage	Fold	ACE	MSB	MCPT	ToB	AgD	ASB	RC	AgH	TE
Bash command injection	L1, L6, A2, A11	A	●	○	○	○	○	◐	◐	○	◐
Credential direct read	L2, A2, A3*	B	●	○	●	◐	○	◐	●	○	◐
Silent exfil via logging	L2, A2	B	●	○	○	◐	○	○	○	○	○
Resource exhaustion	L10, A4	E	●	○	○	○	○	○	◐	○	○
Audit-log poisoning	A1, A8	F	●	○	○	○	◐	◐	○	○	○
Schema-shape tampering	L3	G	●	◐	◐	●	○	○	○	○	○
Metadata prompt injection	L1, A2	C	●	●	●	●	◐	◐	○	○	○
Retrieval reference injection	L1	C	●	●	○	◐	●	◐	◐	○	◐
Output directive write	L5, A7*	D	●	◐	◐	○	◐	○	◐	○	○
Network redirect exfil	L2, A2	B	●	○	◐	◐	◐	◐	●	◐	◐
Preference manipulation	L1, A2, A6*	C	●	●	○	◐	○	◐	◐	○	○
Multi-stage persistence	L1, A1	F	●	○	○	◐	○	◐	◐	○	○

discriminative kernel features. Detector separation lives in the p90 and above tails. Appendix E.5 reports per-feature quantile comparisons.

2.6 Attack-mechanic taxonomy

The same underlying malicious *mechanic*, e.g., "execute `bash -c` following read of payload-bearing content", can arrive through multiple delivery vectors (tampered MCP, poisoned file, poisoned web response, user prompt, poisoned memory file), and prior benchmarks catalog these attacks by heterogeneous axes: delivery vector, attacker-tool label, injection task, CWE, or content-harm category. None of these axes surfaces the shared *observable signature* a runtime sensor would see. We instead organize by *sensor-observable execution mechanic*: the concrete observable action and evidence channel, independent of delivery vector and harm content.

ACE's 17 threat models map onto 12 attack mechanics, grouped into 7 OWASP-aligned held-out folds as shown in Table 1. Mechanics are grouped by shared OWASP anchor, so each fold holds out an OWASP-coherent attack family rather than an arbitrary partition (e.g., fold B groups the `LLM02`-anchored credential-exfiltration mechanics, and fold C the `LLM01`-anchored injection mechanics). Folds are used for the OOD evaluation regime (§3.3). The full threat-model→mechanic→fold mapping and category-by-category OWASP breakdown are in Appendices E.2 and E.6. ACE spans 14 of the 25 OWASP threat categories, including three with partial coverage.

3 Experiments

This section formalizes the detection task, describes how each captured session is featurized, names the detector families we evaluate, and reports results. The *Session Evidence* and *Detection* panels of Figure 1 illustrate the objects introduced here.

3.1 Task formalization

Let a captured session be $s = (\tau_k, \tau_a)$ where τ_k is the in-container kernel-syscall trace of the tool process tree and τ_a is the application-layer evidence (served `tools/list` manifest, user prompt, and agent transcript). A per-patch fire predicate $f : s \to \{0, 1\}$ matches when the sensor observes the attack's effect, using kernel-side syscall signatures (e.g. credential-file `openat` followed by

`sendto` to a synthetic exfil endpoint), app-side signatures (e.g. served tool description contains a poisoned substring, or the agent's response echoes a smuggled directive), or a combinator over both.

A session is benign if no malicious patch was applied. A session is malicious if a malicious patch was applied and the fire predicate matched on the captured evidence ($f(s) = 1$). All other sessions — malicious patch applied but no fire-predicate match — are treated as $y = 0$ at scoring time, except for a small structurally-ambiguous class (fire predicate did not match despite the target tool being exercised on a signature-bearing mechanic) which is dropped from the AUROC cohort. See Appendix E.3 for the full class definitions and scoring policy.

The featurizer produces three view-specific feature vectors $\phi_A(s), \phi_K(s), \phi_C(s)$ where $\phi_C(s) = [\phi_A(s) \,\|\, \phi_K(s)]$ is feature-level concatenation. A view-specific detector $d_V : s \to [0, 1]$ for $V \in \{\text{App}, \text{Kernel}, \text{Cross}\}$ produces a maliciousness score. We report AUROC of d_V against $y(s)$ under two fold designs.

3.2 Featurization and views

Each session is mapped onto an 86-dimensional numeric feature vector: 57 kernel-layer features derived from τ_k (session totals, per-syscall count / log-count / rate across 14 tracepoints, structural syscall-pair ratios, and IP / path cardinalities) and 29 app-layer features derived from τ_a (prompt structural features, schema structural features over the served `tools/list`, tool-call / tool-result trajectory features, and result-novelty features). Features observe structural counts and cardinalities, not identities (no feature reads mechanic-specific sentinel strings by name), so the featurizer is rename-invariant. Full per-feature definitions appear in Appendix G. Every classical and deep-tabular detector reads from this row schema.

The three evaluation views are: App-View (29 features), served protocol surface + trajectory + prompt structure only, what an intent-layer auditor sees; Kernel-View (57 features), in-container syscall trace only, what a kernel-layer host-IDS sees; and Cross-View (86 features), the concatenation of the two, corresponding to $\phi_C(s)$ above. Throughout the paper each view name denotes the evidence slice. When the representation matters, we distinguish the numeric *features* (consumed by the classical and deep-tabular detectors) from the text *rendering* (consumed by the LLM and SLM judges, detailed in Appendix J.3).

LLM and SLM judges receive a view-conditioned text rendering of the same evidence rather than the numeric featurizer. The Kernel-View rendering is a bounded structured (JSON) summary of the strace stream, selected against three alternatives in a representation ablation across three Qwen LLM judges (Appendix J.2), and the winning rendering is used across every LLM and SLM cell in the paper. Per-family input details are in Appendix F.

3.3 Evaluation regimes

Every detector is evaluated under two complementary fold designs on ACE:

- **5-fold stratified (in-distribution, ID).** Standard k-fold on the ACE cohort. Every attack mechanic present in the corpus appears in both the train and test rows of every fold, in proportional counts. This regime reports the ceiling AUROC each detector reaches when the training and test distributions match: a measurement of how well the detector fits the corpus rather than of generalization.
- **7-fold OWASP-aligned (out-of-distribution, OOD).** Each fold holds out one OWASP-anchored attack family (A through G, per Section 2.6). The held-out family is *entirely absent* from the training rows and LLM demonstrations. Matched-benign per host is applied so that benign rows from the same hosts that contributed the held-out attacks are also held out. This regime reports how well a detector generalizes to attack shapes not represented in training: the deployment-relevant number.

AUROC summaries for the ID and OOD evaluations on ACE report the mean across folds with $\pm$ the mean per-fold patch-clustered bootstrap standard error. Each fold uses 1,000 resamples over patch clusters (malicious sessions grouped by attack fixture and benign sessions by host), preserving within-cluster dependence.

3.4 DETECTOR FAMILIES SURVEYED

We evaluate four detector families (Figure 1, *Detection* panel) on identical evidence, plus one off-the-shelf rule-based baseline for reference. *Classical:* AdaBoost and XGBoost. *Deep tabular:* two pretrained tabular foundation models, TabPFN (Hollmann et al., 2025) and TabICL (Qu et al., 2025), that consume the same 86-dimensional feature vector as the classical family and perform in-context learning at inference time: the training rows enter as context and the test rows are scored in a single forward pass, with no gradient training on ACE. Train-from-scratch tabular DL alternatives (MLP, FT-Transformer (Gorishniy et al., 2021), Appendix I.1) and sequence-structured detectors on the raw kernel-syscall stream (LSTM language modelling per Kim et al. (2016), BERT-MIL (Ilse et al., 2018), and Set Transformer variants (Lee et al., 2019), Appendix I.4) both underperform on ACE-shaped data, so we focus primarily on the classical, deep-tabular, LLM-judge, and fine-tuned SLM model families. *LLM judges:* three open-weight models (Qwen3-235B, Qwen3-80B, Qwen3-32B), each evaluated across a 2×2 sweep of prompt template (baseline vs. OWASP-grounded) and shot count ($K{=}0$ zero-shot vs. $K{=}1$ 14-reference few-shot). ID uses fixed references spanning A–G; OOD uses references from each fold's training pool, excluding the test family and test sessions. Model weights remain frozen, and reference sessions are excluded from scoring. This yields 12 (model $\times$ template $\times$ shot) configurations. Figure 2 reports each model under the OWASP-grounded, $K{=}1$ configuration. Appendix J gives the full sweep and reference-selection procedure, and additionally evaluates detection with examples covering all attack families (§J.4). *Fine-tuned SLM judges:* three open models (Qwen2.5-3B/7B, Llama-3.1-8B) LoRA-tuned per (view, fold) with a `malicious`/`benign` verdict token as the target. *Rule-based runtime baseline:* Falco's default rule pack (`falco_rules.yaml`), the canonical open-source runtime rule engine for Linux workloads (The Falco Authors, 2026). We port every rule with a syscall-observable predicate directly against ACE's strace event stream and score sessions by the count of distinct rules that fire. Falco reads syscalls only, so it contributes a Kernel-View score with no App-View or Cross-View. Appendix F summarizes each family's input representation and classical model configurations. Hyperparameters, prompt protocols, ported rules, and sweep grids are in Appendices I–K and L.2.

3.5 RESULTS

Figure 2 reports every detector family side-by-side, with three bars per detector showing the App-, Kernel-, and Cross-Views in each regime. Per-view AUROC cells with SEs are in Table 8 in Appendix H.

Kernel evidence is discriminative. On the Kernel-View alone, every non-linear detector reaches OOD AUROC well above chance: classical XGBoost 0.750 and AdaBoost 0.733, every LLM judge in $[0.725, 0.775]$, and every fine-tuned SLM above 0.88. The paper's strongest OOD result overall, LoRA-fine-tuned Llama-3.1-8B at 0.922, consumes only the Kernel-View. Kernel evidence is discriminative without any application-layer signal.

Cross-View generally improves detection. Combining application and kernel evidence achieves the highest AUROC for all 10 detectors in-distribution and seven of 10 under OOD. For boosted-tree detectors, Cross-View improves OOD AUROC over the best individual view by 10.7 percentage points for AdaBoost ($0.733 \rightarrow 0.840$) and 10.1 percentage points for XGBoost ($0.750 \rightarrow 0.850$). Paired cluster-bootstrap comparisons further show that Cross-View exceeds App-View for all 10 detectors, with each pointwise 95% interval for the AUROC difference above zero (Appendix H.4). The remaining three detectors, Qwen3-32B, Qwen2.5-3B, and Llama-3.1-8B, perform best with Kernel-View under OOD. Appendix K.4 examines these cases.

Kernel evidence generalizes across attack and detector families. Under the seven-fold OWASP-aligned evaluation, all four detector families reach OOD AUROC above 0.80 using kernel evidence: XGBoost Cross-View achieves 0.850, TabPFN Cross-View 0.874, frozen Qwen3-80B with $K{=}1$ Cross-View 0.810, and LoRA-fine-tuned Llama-3.1-8B Kernel-View 0.922. These results demonstrate generalization to held-out attack families across distinct model architectures and training regimes. All three frozen Qwen models benefit from labeled examples drawn from attack families other than the one being tested. Adding these examples to the OWASP-grounded prompt improves Cross-View OOD AUROC by 8.2–20.9 percentage points compared with the same prompt

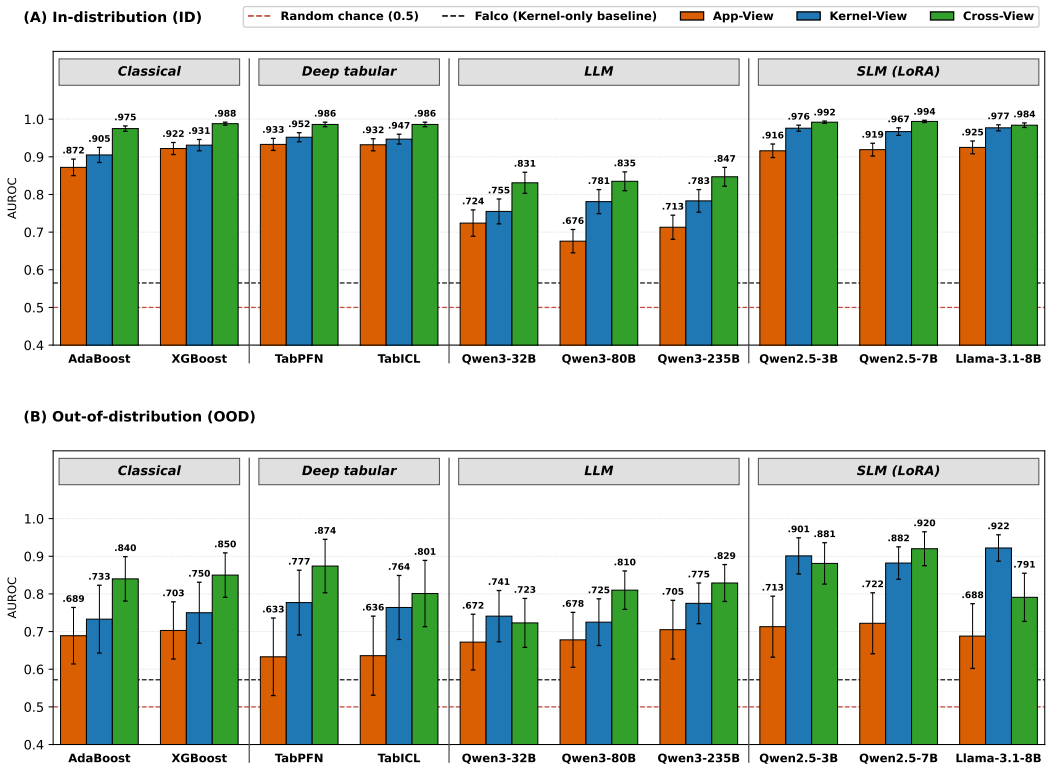


Figure 2: **Unified detector benchmark on ACE.** Mean AUROC across folds for each of 10 detectors and three evidence views (App / Kernel / Cross). **(A)** in-distribution (5 stratified folds); **(B)** out-of-distribution (7 OWASP-aligned held-out folds). Each detector family consumes its assigned view under its own representation (Appendix F). Error bars show the mean per-fold patch-clustered bootstrap SE (§3.3). The dashed red line marks random-chance AUROC (0.5); the dashed black line marks the Falco default-rules baseline (0.565 ID, 0.572 OOD), a Kernel-only reference. LLM-judge rows use the OWASP-grounded prompt with a 14-reference $K{=}1$ protocol, with the test family excluded from OOD demonstrations (Appendix J.3). Per-view AUROC cells with SEs for all detectors are in Table 8 in Appendix H.

without examples (Appendix J.1). Appendix J examines prompt and exemplar sensitivity. Full benchmark results and paired view comparisons appear in Appendix H, with identifier-redaction analyses in Appendix K.

Kernel evidence transfers across agent runtimes. All ACE captures use one agent runtime (`claude-haiku-4-5` inside the Claude Code CLI), so we test transfer on ACE-XA, a 592-session companion corpus captured under Mistral's Devstral 2 123B inside a small Python ReAct scaffold (Yao et al., 2023) with our own stdio MCP client, spanning five OWASP folds (A, B, D, E, F) and four delivery vectors including tampered MCP. Each detector is fit, fine-tuned, or prompted only on ACE fold-X material and scored on ACE-XA fold-X sessions, testing runtime transfer within known attack families. Three of four families clear the discrimination threshold of $\alpha{=}0.75$ under the runtime swap (Hosmer et al., 2013; Mandrekar, 2010): XGBoost Cross-View 0.842 ± 0.099, Qwen3-80B $K{=}1$ Cross-View 0.998 ± 0.002, and Llama-3.1-8B LoRA Kernel-View 0.849 ± 0.097. AUROC is averaged over five folds, with $\pm$ denoting mean per-fold bootstrap SE. TabPFN Cross-View is the exception at 0.662 ± 0.172: permutation importance traces the collapse to a single featurizer field, `path_n_total` (the count of distinct paths beginning with `/tmp`), nonzero on every ACE session because the Claude CLI writes cache and IPC files there but zero on 57% of ACE-XA sessions because our scaffold processes tool results in memory. Full protocol and per-fold results are in Appendix M.

Table 2: **Per-mechanic ID AUROC across 10 detectors.** Mean of per-detector AUROCs computed from pooled held-out ID predictions (5-fold stratified) for the 10 non-Falco detectors of Table 8, under the canonical positive-class labeling (§3.1) with matched-benign-per-host aggregation. ± is the mean patch-clustered bootstrap SE (1,000 resamples, fixed seed) averaged across detectors, not the SE of the mean. Winning view per row in bold (before rounding). *Signal* lists the views (A=App, K=Kernel, C=Cross) clearing the acceptable-discrimination threshold α=0.75 (Appendix H.2) in descending AUROC order; n_+/n_- are per-mechanic positive and matched-negative session counts before detector-specific scoring exclusions.

Mechanic	App	Kernel	Cross	Signal	n_+	n_-
Bash command injection	.812±.051	.846±.046	**.915±.027**	CKA	264	311
Credential direct read	.876±.051	.882±.039	**.978±.012**	CKA	227	529
Silent exfil via logging	.586±.045	**.976±.011**	.970±.010	KC	244	469
Resource exhaustion	.915±.044	.939±.027	**.973±.016**	CKA	109	309
Audit-log poisoning	.608±.109	.932±.049	**.954±.039**	CK	11	108
Schema-shape tampering	.672±.071	.844±.051	**.867±.032**	CK	85	214
Metadata prompt injection	.783±.066	.756±.054	**.902±.038**	CAK	113	133
Retrieval reference injection	.906±.046	.703±.064	**.927±.032**	CA	53	99
Output directive write	.967±.023	.962±.026	**.970±.020**	CAK	162	235
Network redirect exfil	.791±.109	.794±.037	**.871±.042**	CKA	37	43
Preference manipulation	.687±.046	.701±.049	**.834±.028**	C	29	60
Multi-stage persistence	.883±.073	.867±.084	**.982±.013**	CAK	24	63

Signal location varies by attack mechanic. Table 2 decomposes the ID cohort by the 12 attack mechanics of §2.6, labeling a view as carrying *signal* for a mechanic when its mean AUROC across the 10 detectors clears α=0.75. Appendix H.2 documents the threshold's basis, the label methodology, label sensitivity to α, and why this decomposition is reported in-distribution. Which single layer carries that signal shifts by mechanic. Adding kernel evidence to the application layer improves detection on all 12 mechanics, most on Silent exfil via logging (+.384) and Audit-log poisoning (+.347). Cross-View is the only view clearing α on all 12, and wins on 11. Kernel-View wins on Silent exfil via logging, exceeding Cross by .006. On Output directive write, Cross exceeds App by .003. These descriptive differences are small relative to the reported per-detector uncertainty. The decomposition therefore supports composition as a reliable choice across the taxonomy without a uniform improvement on every mechanic. The layers are complementary: App falls below α on three mechanics that Kernel resolves, while Kernel falls below it on Retrieval reference injection, which App resolves. Silent exfil via logging is weak on App-View (.586) and the corpus's strongest kernel cell (.976), because neither the credential read nor its write to a log sink ever surfaces in the transcript. Retrieval reference injection inverts this (App .906, Kernel .703): the attack's whole effect is a text-surface manipulation carrying no distinctive syscall footprint. Preference manipulation is the one mechanic no single layer resolves (.687 App, .701 Kernel); composition clears both α=0.75 and 0.80 (.834 ± .028).

4 Conclusion

Every detector's strongest configuration in Figure 2 consumes kernel signal, and off-the-shelf rule engines calibrated for server workloads are near-random on interactive-agent workloads. On the strength of these results we recommend that future approaches to agent security consider kernel-level evidence as a first-class input to detection, alongside the application-layer signals the field has historically focused on. Remaining open problems, limitations, and broader-impact discussion appear in Appendices B–D. We will release ACE as an open benchmark for the community to explore and extend this direction.

Reproducibility statement

The corpus construction protocol, threat-model taxonomy, and positive-class labeling policy are documented in §3.1 and Appendix E, and the fold designs and bootstrap procedure in §3.3. The appendices describe the featurizer, detector configurations, prompt protocols, fine-tuning hyperparameters, and detailed evaluation results. Upon acceptance we will publicly release the ACE corpus,

source code for featurization, training, and evaluation, exact prompt templates, split definitions, trained detector weights, and scripts that regenerate the reported results.

ETHICS STATEMENT

The corpus contains adversarial patches against real MCP servers. Every patch is air-gapped by construction: synthetic credentials replace real secrets, IETF-reserved TEST-NET ranges (`192.0.2.0/24`, `198.51.100.0/24`, `203.0.113.0/24`) replace real exfiltration endpoints, loopback listeners replace real network sinks, and a static validator rejects any patch referencing a non-fixture credential path or non-reserved IP. Release of the corpus is a dual-use contribution: it enables defensive research on agent-runtime observability but also documents realistic attack patterns. We argue the balance favors release: the mechanisms we catalog are already publicly disclosed (OSV records (Google Open Source Security Team, 2024), CVEs, industry blog posts), and defenders currently lack a paired benign/malicious dataset to build detectors against.

USE OF LARGE LANGUAGE MODELS

The authors acknowledge the use of large language models, employed exclusively to assist with grammar correction, proofreading, and minor stylistic refinements throughout the manuscript. The use of LLMs was strictly limited to language polishing and did not contribute to the research content itself. All core research, methodology, experimental design, and substantive findings represent the original work of the authors, who retain full responsibility for the content and conclusions presented.

REFERENCES

Baichao An, Pei Chen, Geng Hong, Yueyue Chen, and Mengying Wu. FlowGuard: From Signals to Evidence for MCP Security Detection. https://arxiv.org/abs/2607.14754, 2026.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, J. Zico Kolter, Matt Fredrikson, Eric Winsor, Jerome Wynne, Yarin Gal, and Xander Davies. AgentHarm: A Benchmark for Measuring Harmfulness of LLM Agents. In *International Conference on Learning Representations (ICLR)*, 2025. https://arxiv.org/abs/2410.09024.

Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, 2024. November 2024 launch.

Anthropic. Mapping AI-enabled Cyber Threats: Insights from the LLM ATT&CK Navigator. https://www.anthropic.com/research/attack-navigator, 2026. Frontier Red Team, accessed 2026-06-30.

Aqua Security. Tracee: Linux runtime security and forensics using ebpf. https://github.com/aquasecurity/tracee, 2026. accessed 2026-06-17.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. https://arxiv.org/abs/2005.14165.

Christoph Bühler, Matteo Biagiola, Luca Di Grazia, and Guido Salvaneschi. AgentBound: Securing Execution Boundaries of AI Agents. In *Proceedings of the 34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2026. https://arxiv.org/abs/2510.21236.

Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance. In *IEEE Symposium on Security and Privacy*, 2024. https://arxiv.org/abs/2308.05034.

Cisco AI Defense. Cisco MCP Scanner: security analysis for Model Context Protocol servers. https://github.com/cisco-ai-defense/mcp-scanner, 2026. Open-source YARA + LLM-as-judge scanner for MCP servers and tools; Apache-2.0.

Gideon Creech and Jiankun Hu. ADFA Intrusion Detection Datasets (ADFA-LD). https://research.unsw.edu.au/projects/adfa-ids-datasets, 2013. University of New South Wales, accessed 2026-06-15.

Cybersecurity and Infrastructure Security Agency (CISA). SUNBURST: SolarWinds Orion supply-chain attack. https://www.cisa.gov/news-events/alerts/2020/12/13/active-exploitation-solarwinds-software, 2020. trojanized build of SolarWinds Orion network management platform.

DARPA I2O. DARPA Transparent Computing Engagement Materials. https://github.com/darpa-i2o/Transparent-Computing, 2020. accessed 2026-06-15.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating Prompt Injections by Design. https://arxiv.org/abs/2503.18813, 2025.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. https://arxiv.org/abs/2305.14314.

Gloria Felicia, Zitha Sasindran, Jinfeng He, Michael Eniolade, Hemant Kumar, and Milan Hussain Angati. StepShield: When, Not Whether to Intervene on Rogue Agents. *arXiv preprint arXiv:2601.22136*, 2026. https://arxiv.org/abs/2601.22136.

Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A Sense of Self for Unix Processes. In *IEEE Symposium on Security and Privacy*, pp. 120–128, 1996. doi: 10.1109/SECPRI.1996.502675. https://ieeexplore.ieee.org/document/502675.

GitHub Advisory Database. ua-parser-js npm package compromise. https://github.com/advisories/GHSA-pjwm-rvh2-c87w, 2021. cryptominer and credential stealer in versions 0.7.29, 0.8.0, 1.0.0.

Google. Function calling with the Gemini API (Remote MCP integration). https://ai.google.dev/gemini-api/docs/function-calling, 2025. Gemini API documents Remote MCP support; MCP integration announced April 2025.

Google Open Source Security Team. OSV: A Distributed Vulnerability Database for Open Source. https://osv.dev, 2024. Accessed 2026-07.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems*, 2021. https://proceedings.neurips.cc/paper/2021/hash/9d86d83f925f2149e9edb0ac3b49229c-Abstract.html.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. https://doi.org/10.1145/3605764.3623985.

Martin Grimmer, Martin Max Röhling, Dennis Kreußel, and Simon Ganz. A modern and sophisticated host based intrusion detection data set. In *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung (16. Deutscher IT-Sicherheitskongress)*, pp. 135–145, 2019. LID-DS dataset introduction, https://dbs.uni-leipzig.de/research/publications/a-modern-and-sophisticated-host-based-intrusion-detection-data-set; dataset and 2021 revision at https://github.com/LID-DS/LID-DS.

Chengquan Guo, Xun Liu, Chulin Xie, Andy Zhou, Yi Zeng, Zinan Lin, Dawn Song, and Bo Li. RedCode: Risky Code Execution and Generation Benchmark for Code Agents. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. https://arxiv.org/abs/2411.07781.

Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *Network and Distributed System Security Symposium (NDSS)*, 2020. doi: 10.14722/ndss.2020.24046. https://arxiv.org/abs/2001.01525.

Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *Network and Distributed System Security Symposium (NDSS)*, 2019. doi: 10.14722/ndss.2019.23349.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 2025. https://www.nature.com/articles/s41586-024-08328-6.

David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. John Wiley & Sons, 3rd edition, 2013. ISBN 978-0-470-58247-3. doi: 10.1002/9781118548387. https://doi.org/10.1002/9781118548387.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2022. https://arxiv.org/abs/2106.09685.

Maximilian Ilse, Jakub M. Tomczak, and Max Welling. Attention-based Deep Multiple Instance Learning. In *International Conference on Machine Learning (ICML)*, 2018. https://arxiv.org/abs/1802.04712.

Invariant Labs. MCP Security Notification: Tool Poisoning Attacks. https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks, 2025. Invariant Labs Blog, accessed 2026-06-15.

Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, and Sungroh Yoon. LSTM-Based System-Call Language Modeling and Robust Ensemble Method for Designing Host-Based Intrusion Detection Systems. *arXiv preprint arXiv:1611.01726*, 2016. https://arxiv.org/abs/1611.01726.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *International Conference on Machine Learning (ICML)*, 2019. https://arxiv.org/abs/1810.00825.

Dawei Li, Yuguang Yao, Zhen Tan, Huan Liu, and Ruocheng Guo. ToolPRMBench: Evaluating and Advancing Process Reward Models for Tool-using Agents. *arXiv preprint arXiv:2601.12294*, 2026. https://arxiv.org/abs/2601.12294.

Jayawant N. Mandrekar. Receiver Operating Characteristic Curve in Diagnostic Test Assessment. *Journal of Thoracic Oncology*, 5(9):1315–1316, 2010. https://doi.org/10.1097/JTO.0b013e3181ec173d.

Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016. https://dl.acm.org/doi/10.1145/2939672.2939783.

Rahul Marchand, Art O Cathain, Jerome Wynne, Philippos Maximos Giavridis, Stuart Jennings, Freddy Tuxworth, Tolga H. Dur, Sam Deverett, John Wilkinson, Jason Gwartz, and Harry Coppock. Quantifying Frontier LLM Capabilities for Container Sandbox Escape. https://arxiv.org/abs/2603.02277, 2026.

Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Benjamin Feuer, Chinmay Hegde, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When Do Neural Nets Outperform Boosted Trees on Tabular Data? In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023. https://arxiv.org/abs/2305.02997.

Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V. N. Venkatakrishnan. POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019a. https://arxiv.org/abs/1910.00056.

Sadegh M. Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V. N. Venkatakrishnan. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In *2019 IEEE Symposium on Security and Privacy (S&P)*, 2019b. https://arxiv.org/abs/1810.01594.

Vijitha Mittapalli, Shreyaa Jayant Dani, Satya Srujana Pilli, Snigdha Ansu, Mohammadreza Teymoorianfard, Franck Dernoncourt, Hongjie Chen, Yu Wang, Ryan A. Rossi, and Nesreen K. Ahmed. TRACE: Trajectory Reasoning through Adaptive Cross-Step Evidence Aggregation for LLM Agents. *arXiv preprint arXiv:2606.07054*, 2026. https://arxiv.org/abs/2606.07054.

Om Narayan, Rashmi Jyoti, and Ramkinker Singh. ChainWatch: A Kill Chain-Aligned Sequential Detection Framework for Multi-Step Attacks in MCP-Based AI Agent Systems. *arXiv preprint arXiv:2607.19432*, 2026. https://arxiv.org/abs/2607.19432.

NIST National Vulnerability Database. CVE-2024-3094: xz-utils backdoor. https://nvd.nist.gov/vuln/detail/CVE-2024-3094, 2024. sshd authentication bypass via tampered upstream tarball.

npm Inc. The event-stream npm package compromise. https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident, 2018. malicious payload targeting Copay wallet; 2M weekly downloads at time of compromise.

OpenAI. New tools and features in the Responses API. https://openai.com/index/new-tools-and-features-in-the-responses-api/, 2025. 21 May 2025; adds remote MCP server support to the Responses API. MCP-based ChatGPT apps followed on 6 October 2025 (https://openai.com/index/introducing-apps-in-chatgpt/).

OWASP Gen AI Security Project. Agentic AI: Threats and Mitigations. https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/, 2025. Version 1.0, February 2025. Archived document: https://www.aigl.blog/content/files/2025/04/Agentic-AI---Threats-and-Mitigations.pdf.

OWASP Gen AI Security Project. OWASP Top 10 for Agentic Applications, 2026. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/, 2026. accessed 2026-06-15.

PulseMCP. PulseMCP: Directory of Model Context Protocol servers. https://www.pulsemcp.com/servers, 2026. accessed 2026-06-30; lists 20,000+ public MCP servers.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. In *International Conference on Machine Learning (ICML)*, 2025. https://arxiv.org/abs/2502.05564.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *International Conference on Learning Representations (ICLR)*, 2024. https://arxiv.org/abs/2309.15817.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. https://arxiv.org/abs/2302.04761.

Rohan Sequeira, Stavros Damianakis, Umar Iqbal, and Konstantinos Psounis. Agent-Sentry: Bounding LLM Agents via Execution Provenance. https://arxiv.org/abs/2603.22868, 2026.

Aditya Sinha, Akshat Naik, Victor Gillioz, Simon Storf, Kilian Merkelbach, Rich Barton-Cooper, Axel Højmark, and Marius Hobbhahn. Training Deliberative Monitors for Black-Box Scheming Detection. *arXiv preprint arXiv:2605.29601*, 2026. https://arxiv.org/abs/2605.29601.

Snyk (formerly Invariant Labs). Snyk agent-scan (formerly Invariant Labs mcp-scan). https://github.com/snyk/agent-scan, 2026. Open-source MCP manifest and tool-description scanner; Apache-2.0.

The Falco Authors. Falco: Cloud-native runtime security. https://falco.org/, 2026. accessed 2026-06-17.

Trail of Bits. Jumping the Line: How MCP Servers Can Attack You Before You Ever Use Them. https://blog.trailofbits.com/2025/04/21/jumping-the-line-how-mcp-servers-can-attack-you-before-you-ever-use-them/, 2025. Trail of Bits Blog, accessed 2026-06-15.

Peiran Wang, Ying Li, and Yuan Tian. Aligning Provenance with Authorization: A Dual-Graph Defense for LLM Agents. https://arxiv.org/abs/2605.26497, 2026.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers. https://arxiv.org/abs/2508.14925, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155*, 2023. https://arxiv.org/abs/2308.08155.

Wenpeng Xing, Zhonghao Qi, Yupeng Qin, Yilin Li, Caini Chang, Jiahui Yu, Changting Lin, Zhenzhen Xie, and Meng Han. MCP-Guard: A Multi-Stage Defense-in-Depth Framework for Securing Model Context Protocol in Agentic AI. https://arxiv.org/abs/2508.10991, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*, 2023. https://arxiv.org/abs/2210.03629.

Sultan Zavrak. Content-Aware Attack Detection in LLM Agent Tool-Call Traffic: An Empirical Study of Features, Architectures, and Evaluation Protocols. *arXiv preprint arXiv:2605.11053*, 2026. https://arxiv.org/abs/2605.11053.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics (ACL Findings)*, 2024. https://aclanthology.org/2024.findings-acl.624/.

Dongsen Zhang, Zekun Li, Xu Luo, Xuannan Liu, Peipei Li, and Wenjun Xu. MCP Security Bench (MSB): Benchmarking Attacks Against Model Context Protocol in LLM Agents. https://arxiv.org/abs/2510.15994, 2025a.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. In *International Conference on Learning Representations (ICLR)*, 2025b. https://arxiv.org/abs/2410.02644.

Yusheng Zheng, Yanpeng Hu, Tong Yu, and Andi Quinn. AgentSight: System-Level Observability for AI Agents Using eBPF. https://arxiv.org/abs/2508.02736, 2025.

Yusheng Zheng, Tianyuan Wu, Quanzhi Fu, Tong Yu, Wenan Mao, Tao Ma, Dan Williams, Wei Wang, and Andi Quinn. ActPlane: Programmable OS-Level Policy Enforcement for Agent Harnesses. https://arxiv.org/abs/2606.25189, 2026.

CONTENTS

A Related work **17**

A.1 Agent-attack catalogs 17

A.2 Intent-layer agent defenses 17

A.3 Learned monitoring of agent trajectories 17

A.4 Host intrusion-detection corpora 18

B Limitations **18**

C Future work **19**

D Broader impact **19**

E Full corpus composition **19**

E.1 Threat model details 20

E.2 Threat model → mechanic → fold mapping 20

E.3 Session labels and fold design 21

E.4 Positive-class robustness check 22

E.5 Benign class legitimacy check 23

E.6 OWASP coverage detail 24

E.7 Capture pipeline detail 24

E.8 Session-level attribution validation 26

E.9 Safety perimeter 27

F Detector inputs **28**

G Feature list **28**

G.1 Kernel-Layer features 29

G.2 App-Layer features 29

G.3 Design choices 30

H Detector benchmark details **30**
H.1 Full per-cell benchmark . . . 30
H.2 Per-mechanic signal classification . . . 30
H.3 Sensitivity to captured user prompts . . . 32
H.4 Paired comparisons between evidence views . . . 32

I Deep tabular details **33**
I.1 Tabular DL ablation . . . 33
I.2 MLP hyperparameter configuration . . . 34
I.3 FT-Transformer hyperparameter configuration . . . 35
I.4 Sequence-DL ablation on the raw kernel stream . . . 35

J LLM details **36**
J.1 LLM prompt and exemplar ablation . . . 36
J.2 Kernel-View feature representation ablation . . . 38
J.3 LLM prompt templates . . . 39
J.4 Detection with examples of known attack families . . . 40

K SLM details **41**
K.1 LoRA training recipe . . . 41
K.2 Featurized text rendering per view . . . 42
K.3 Reproducibility . . . 43
K.4 Discussion . . . 43
K.5 Identifier redaction in kernel summaries . . . 43

L Comparison with standard detection tools **44**
L.1 Pre-deployment MCP scanner (Cisco mcp-scanner) . . . 44
L.2 Runtime rule engine (Falco) . . . 47
L.3 Discussion . . . 48

M Cross-agent generalization ablation **48**
M.1 ACE-XA corpus construction . . . 48
M.2 Evaluation protocol . . . 49
M.3 Per-fold results . . . 50
M.4 Reading: why TabPFN under-transfers . . . 51

N Compute cost **51**
N.1 Session capture . . . 51
N.2 LLM-judge sweep . . . 51
N.3 SLM LoRA training . . . 52
N.4 Classical, deep-tabular, and rule-based detectors . . . 52

N.5 Aggregate 52
N.6 Cross-agent and exemplar-selection studies 53

A Related work

This appendix situates our contribution against three research lines that bracket it: agent-attack catalogs, intent-layer defenses, and host intrusion-detection corpora.

A.1 Agent-attack catalogs

Attack-side benchmarks span protocol tampering, injection, consequence, and harm framings. *MSB* (Zhang et al., 2025a) benchmarks twelve attack categories across 405 tools and 2,000 instances; *MCPTox* (Wang et al., 2025) embeds malicious instructions in tool metadata across 45 live MCP servers and 1,312 test cases; *AgentDojo* (Debenedetti et al., 2024) is a dynamic prompt-injection evaluation environment; *InjecAgent* (Zhan et al., 2024) benchmarks indirect prompt injection (Greshake et al., 2023) with 1,054 test cases; and *ASB* (Zhang et al., 2025b) catalogs agent security scenarios by delivery vector. Consequence- and harm-shaped alternatives include *RedCode* (Guo et al., 2024) (CWE-style code-execution outcomes), *AgentHarm* (Andriushchenko et al., 2025) (content-harm categories), and *ToolEmu* (Ruan et al., 2024) (sandboxed tool-behavior emulation). Industry disclosures from Trail of Bits (Trail of Bits, 2025) and Invariant Labs (Invariant Labs, 2025) expand the threat landscape, and OWASP's 2026 Agentic Top 10 (OWASP Gen AI Security Project, 2026) situates these patterns within a broader threat-model framework. All operate at the protocol and tool-call layer: none capture kernel execution traces or release paired benign/malicious sessions from the same tool surface under the same prompt.

A.2 Intent-layer agent defenses

CaMeL (Debenedetti et al., 2025) extracts control and data flow from the user query and enforces capability-style policies on tool invocations; *MCP-Guard* (Xing et al., 2025) combines static scanning, a fine-tuned semantic detector, and an LLM arbitrator; Anthropic's Attack Navigator (Anthropic, 2026) catalogs attack patterns at the agent–tool interface. Pre-deployment MCP scanners — Cisco `mcp-scanner` (Cisco AI Defense, 2026) and Snyk `agent-scan` (Snyk (formerly Invariant Labs), 2026) — inspect a server's advertised metadata or source before any agent session runs. Complementary systems operate at the runtime layer with different guarantees: *FlowGuard* (An et al., 2026) enforces information-flow policies over tool invocations, and *AgentBound* (Bühler et al., 2026) implements capability-based access control at the agent–tool interface. All of these operate above the OS boundary: a tool call that passes every prompt-layer guard and presents a clean schema can still, when it runs, read `~/.aws/credentials` and exfiltrate to an attacker endpoint. Pre-deployment scanners also have no visibility into ACE's five non-MCP delivery-vector families (51% of the corpus). Running Cisco `mcp-scanner` even on the tampered-MCP subset yields near-random recall at high FPR, while all four ACE detector families achieve higher recall and lower FPR at their native thresholds on the same subset (Appendix L).

A.3 Learned monitoring of agent trajectories

A separate line of work builds learned models that read the agent's tool-call trajectory directly. These monitors consume the sequence of (`tool_name`, `arguments`, `result`) tuples produced during execution and score maliciousness or misalignment per-step or per-trajectory. *TRACE* (Mittapalli et al., 2026) uses a triage-inspect-judge loop over long-horizon trajectories to detect hidden malicious behavior on SHADE-Arena. *ChainWatch* (Narayan et al., 2026) applies a hidden Markov model over MCP tool-call sequences aligned to a six-stage kill chain. Zavrak's *content-aware MCP tool-call detector* (Zavrak, 2026) embeds tool descriptions and results with a sentence encoder and compares GNN, MLP, and tree-ensemble scorers over the resulting session graphs. *ToolPRMBench* (Li et al., 2026) formalises a step-level process-reward-model benchmark for tool-using agents. *Deliberative monitors* (Sinha et al., 2026) distill scheming-detection ability from frontier teachers into smaller open-weight action-only classifiers, the design most directly analogous to our LoRA-fine-tuned SLM judges. *StepShield* (Felicia et al., 2026) introduces an Early-Intervention-

Rate metric that scores rogue-agent monitors on *when* they detect harm rather than only *whether* they do. All of these operate on the same App-View evidence stream ACE captures at the protocol surface, namely served tool schemas, tool invocations, and tool results. None pairs that stream with a kernel-side syscall trace on the same sessions, so a comparison between application-layer trajectory monitors and kernel-layer detectors, the central question of this paper, has not been possible on a common substrate.

A.4 Host intrusion-detection corpora

The OS-layer host-IDS lineage long predates the agent era. STIDE (Forrest et al., 1996) models a process's normal behavior as the distribution of short syscall sequences and seeded decades of syscall-trace anomaly detection benchmarked against ADFA-LD (Creech & Hu, 2013) and LID-DS (Grimmer et al., 2019); provenance-graph detectors (Manzoor et al., 2016; Han et al., 2020; Cheng et al., 2024; Milajerdi et al., 2019b;a; Hassan et al., 2019) trained on DARPA Transparent Computing traces (DARPA I2O, 2020) extend to APT scenarios; production EDRs (The Falco Authors, 2026; Aqua Security, 2026) score against syscall, file, and network-cardinality rules. None of these corpora or platforms contains LLM-agent context, MCP servers, or paired benign/malicious sessions. On the agent side, *AgentSight* (Zheng et al., 2025) and *ActPlane* (Zheng et al., 2026) bridge eBPF boundary tracing to agent prompts (AgentSight combines it with an LLM analyzer; ActPlane enforces OS-level policies), and Agent-Sentry (Sequeira et al., 2026), AuthGraph (Wang et al., 2026), FlowGuard (An et al., 2026), and AgentBound (Bühler et al., 2026) instrument tool-call provenance or MCP-specific runtime evidence at the application layer. All are enforcement or observability systems. None releases a paired-session corpus that would let a Cross-View detector be evaluated on a matched benign / malicious control.

B Limitations

Six limitations bound the interpretation of the results in this paper.

(i) The two-layer sensor is blind to three attack-mechanic classes. Pure tool-return content manipulation, inter-call timing channels, and whitespace steganography produce no signature on either evidence layer by construction. These are cataloged as scope-boundary demonstrations, not detection targets, and would require additional sensors (output-content classifiers, timing monitors) to become detectable.

(ii) Once-per-session protocol-surface snapshot. The served `tools/list` manifest is captured once at session start. An attacker who mutates the served manifest mid-session falls outside this snapshot. Runtime protocol mutation is a sensor-coverage gap for the current design.

(iii) Session-level attribution. Session-level attribution is deterministic under our capture design (§2.4), but sub-session per-tool-call attribution is a harder open problem out of scope for this work.

(iv) Capture-environment network shape. All non-loopback traffic in ACE is intercepted by an in-lab sinkhole (Appendix E.7), so attacks targeting the IETF-reserved TEST-NET exfiltration addresses (Appendix E.9) succeed at the syscall layer with `connect(...) = 0` and `sendto(...) = <bytes>` rather than failing with `-1 ENETUNREACH`. The network-side syscall fingerprint ACE captures — including the `connect`, `sendto`, `pair_connect_sendto_ratio`, and `ip_external_count` features — is therefore the shape produced by a sinkholed connection, which differs from the shape a real-deployment capture would record for the same attack.

(v) Compute cost of reproduction. ACE session capture and ID/OOD detector evaluation (three-model Qwen prompt and reference comparisons, and SLM LoRA training) cost approximately $1,464–$8,545 at published Bedrock and `p4d.24xlarge` rates listed in Appendix N, depending on SLM training parallelism. This cost is substantial, and full reproduction or extension of these evaluations at a comparable scale may be difficult for smaller research budgets. Practitioners exploring a subset (e.g., a single detector family or a single held-out fold) can reduce cost by an order of magnitude while retaining the paper's headline comparisons. Releasing the detector weights, LoRA adapters, and per-fold score files will eliminate the need to re-run the training and inference sweeps for downstream analysis. Appendix N reports the per-component breakdown, per-model token esti-

mates, and the parallel versus serial SLM accounting. The additional transfer and exemplar studies are accounted for separately there.

(vi) LLM-judge and transfer coverage. The reported frozen-judge sweep covers three Qwen configurations. The Qwen3-80B support study retains aggregate performance across three random exemplar draws under a fixed-size, category-balanced protocol. Broader support designs and LLM providers remain untested. The cross-agent study covers four detector families on one alternate runtime, with three retaining mean AUROC above 0.75. Performance varies by mechanic, and Llama's E/F checkpoint-selection sets each contain only two positives. Broader runtime coverage remains future work.

C FUTURE WORK

Four concrete threads follow from the limitations of Appendix B.

Extend the sensor surface. Output-content classifiers and timing monitors would make the three signature-free mechanic classes (pure tool-return manipulation, inter-call timing channels, whitespace steganography) detectable, closing the primary scope-boundary gap of this work.

Corpus expansion along the attack axis. Attack disclosures continue to broaden as the agent-runtime ecosystem matures, and folding newly disclosed mechanic classes into ACE would extend the taxonomy of Table 1 and let the Cross-View detection finding be re-measured on a richer attack surface. Corpus growth along this axis is a natural extension of this work.

Attribution beyond single-session containers. ACE's attribution guarantee relies on a bounded process-descent scope — one `strace` attached to a single root process inside a container's PID namespace (§2.4). Three deployment regimes fall outside this scope: bare-process agents on shared hosts launched without a wrapper, long-running agent daemons that predate sensor attach, and agents embedded in IDEs or desktop applications. Extending attribution to these regimes requires alternative sensor-placement designs — per-process eBPF probes tagged with cgroup or PID-namespace identity at exec-time, ptrace-based late attach, or IDE-plugin instrumentation — each with its own precision and coverage trade-offs relative to the strace-per-container capture used here.

Corpus expansion along the runtime axis. ACE covers the MCP-server and built-in-tool ecosystem that dominates the mid-2026 agent-runtime landscape. Extending the corpus to other agent-runtime shapes — plug-in architectures beyond MCP, multi-agent orchestration frameworks, and other emerging runtimes — would broaden the empirical base for the Cross-View finding and surface any new mechanic classes those architectures introduce.

D BROADER IMPACT

The observation channels this paper studies (eBPF-based syscall tracing and served-schema capture) are already deployment-ready in production runtimes. Building discriminative detectors on that evidence is the direction this paper's results support. The dual-use risk of publishing an attack corpus is real, but every mechanic we catalog is already publicly disclosed in industry advisories, OSV records (Google Open Source Security Team, 2024), and CVEs. The corpus's value to defenders (a paired benign/malicious dataset that did not previously exist) outweighs the marginal information given to attackers. The corpus will be released with attack fixtures air-gapped by construction (synthetic credentials, IETF-reserved TEST-NET range IPs, loopback-bound listeners) so replaying a session against real infrastructure requires deliberate hostile adaptation rather than direct reuse.

E FULL CORPUS COMPOSITION

This appendix presents the full mapping (Table 3) from ACE's 17 threat models onto the 12 attack mechanics of Table 1, along with the pure-negative categories held only as $y = 0$ rows and the per-fold session counts across both fold designs. Section E.1 expands the threat model summarized in §2.1.

E.1 THREAT MODEL DETAILS

Attacker capability. We consider three overlapping attacker models, all represented in the corpus. *(i) Package tampering:* an attacker controls a third-party MCP server package and has it installed by the victim from an unaudited public index (npm, PyPI, GitHub). The compromise mechanism is left abstract: a typo-squatted dependency, a maintainer-account takeover, a post-distribution patch, or a hidden post-install hook. *(ii) Indirect injection against built-in tools:* the attacker plants malicious content in a file, web resource, retrieval corpus, or persistent-memory file that the agent subsequently reads. The tool package itself is intact. The attack rides on the content the agent chooses to consume. *(iii) User-direct malicious prompts:* the user's own prompt asks the agent to perform harmful actions under a benign-looking cover task.

After the initial delivery step, the attacker inherits whatever capabilities the compromised or misdirected tool was granted: typically full user-level access to the local filesystem, network egress, and the ability to spawn child processes.

Detection surface. The defender observes a single session: the agent's transcript (user prompts, tool calls, tool results), the served `tools/list` manifest from each running MCP server (once per session), and the kernel-side syscall trace of the agent's tool processes. The syscall trace is captured by in-container instrumentation attached to the tool processes' subtree.

Defender capability. The defender is a post-hoc auditor: sessions are captured, logged, and scored after they complete. This matches the deployment shape of production EDR and audit-streaming systems.

Scope. We address single-agent single-session detection. Cross-session state, multi-agent coordination, and inline blocking are out of scope. The protocol-surface sensor captures `tools/list` once at session start. An attacker who mutates the served manifest mid-session falls outside this snapshot. Three attack-mechanic classes (pure tool-return manipulation, inter-call timing channels, whitespace steganography) leave no signature on either evidence layer by construction and are cataloged as scope-boundary demonstrations rather than detection targets.

E.2 THREAT MODEL → MECHANIC → FOLD MAPPING

Table 3: **The 17 threat models in ACE mapped onto the 12 attack mechanics and 7 OWASP-aligned held-out folds.** `mcp_tool_tampering` contributes fixtures to every fold; the 16 built-in-tool threat models each map to a single mechanic. Three built-in threat models captured zero malicious sessions (agent-side refusal ceiling) and are kept in every training pool as pure-negatives.

Threat model (delivery vector)	Mechanic	OWASP	Fold
MCP tool tampering	All 12 mechanics	(all)	A–G
File → bash injection (built-in)	Bash command injection	LLM01+06	A
Web → bash injection (built-in)	Bash command injection	LLM01+06	A
User-direct → bash injection (built-in)	Bash command injection	LLM01+06	A
Memory → bash injection (built-in)	Bash command injection	LLM01+06	A
File → credential exfil (built-in)	Credential direct read	LLM02	B
User-direct → credential exfil (built-in)	Credential direct read	LLM02	B
Memory → credential exfil (built-in)	Credential direct read	LLM02	B
Web → exfil redirect (built-in)	0 fires (refusal)	LLM02	pool
Retrieval → reference injection (built-in)	Retrieval reference injection	LLM01	C
File → write directive (built-in)	Output directive write	LLM05	D
User-direct → write directive (built-in)	Output directive write	LLM05	D
Memory → write directive (built-in)	Output directive write	LLM05	D
Bash loop resource abuse (built-in)	Resource exhaustion	LLM10	E
Recursive-fetch resource abuse (built-in)	Resource exhaustion	LLM10	E
Multi-step web → file (built-in)	0 fires (refusal)	LLM01 chain	pool
Multi-step file → web (built-in)	0 fires (refusal)	LLM01 chain	pool

E.3 SESSION LABELS AND FOLD DESIGN

Fire predicate structure. Each malicious patch ships a `signature.json` file alongside its source code, declaring the predicate that must match for the session to be labeled *malicious*. Predicates are composed from atomic clauses that read either the kernel-side trace (τ_k) or the application-layer evidence (τ_a). *Kernel clauses* (read τ_k) check syscall-name matches (e.g., `openat` on a specific path prefix, `connect` to a specific endpoint, `execve` of a shell), argument substrings (path or IP prefixes), and count thresholds (e.g., `execve` count ≥ 3). *App clauses* (read τ_a) check substring matches in a served tool description, substring matches in an agent response transcript, and structural checks on the `tools/list` manifest (new schema field added or new tool added vs. baseline). Atomic clauses are combined under `all` (AND) and `any` (OR) combinators. Of ACE's 141 malicious patch designs, 57 use at least one app-layer clause, 70 are kernel-only, and 14 are signature-free by construction (no fire predicate). The corresponding fire predicate operates over s rather than τ_k alone.

Session labels. Each captured session carries one of four labels applied at scoring time. Benign sessions run the unmodified tool and are $y{=}0$ by construction. Malicious sessions run a malicious variant *and* the corresponding patch-specific fire predicate matched on the captured evidence (e.g., for a credential-exfil variant, a credential-file `openat` followed by network `sendto` to a synthetic exfil endpoint). These are the $y{=}1$ positives. Dormant sessions run a malicious variant but the agent never exercised the target tool (the patched code did not run). These are labeled $y{=}0$. Latent sessions run a malicious variant, the target tool was exercised, but the fire predicate did not match, either because the patch belongs to a signature-free mechanic class (three attack-mechanic categories — `output_manipulation`, `timing_covert`, `output_steg` — plus any patch declared `primary_signal: NONE` produce no signature on either evidence layer by construction and enter every training pool only as $y{=}0$ pure negatives) or because the exploit conditions were not quite met at runtime. The former (signature-free latent, 66 sessions) are labeled $y{=}0$; the latter (signal-bearing latent, 396 sessions) are dropped from scoring as structurally ambiguous.

Non-firing mechanics kept as pure-negatives. Six category groupings enter every training pool as $y = 0$ pure-negatives because their fire rubric is unsatisfiable on the two-layer sensor surface, either by design (three sensor blind spots: pure tool-return manipulation, whitespace steganography, inter-call timing channels) or by observed outcome (three refusal-ceiling threat models: the attacker-URL exfil redirect and the two multi-step chains, where the agent declines the redirected action). These rows are never held out as any fold's test set.

Fold designs. Two fold designs are used across the paper. 5-fold stratified partitions the 4,047-session corpus into approximately equal-size folds while preserving each mechanic's proportion within every train/test split, so every attack mechanic appears in both the train and test rows of every fold. This regime measures how well a detector fits the corpus (ID). 7-fold OWASP-aligned groups the 12 mechanics into 7 held-out families anchored on OWASP LLM Top 10 and Agentic AI Threats and Mitigations categories (Table 1 lists each mechanic's fold letter). Each fold holds out one family entirely and applies matched-benign per host: benign rows from the same MCP servers or built-in-tool hosts that contributed the held-out attacks are also held out. This regime measures how well a detector generalizes to attack shapes not represented in training (OOD).

Per-fold session-label counts. Every session in the 4,047-session corpus carries one of the four labels defined in the label-semantics paragraph above: Benign, Malicious, Dormant, or Latent. Under the paper's positive-class convention, Malicious sessions are $y{=}1$ positives; Benign, Dormant, and signature-free Latent sessions are $y{=}0$ negatives; and signal-bearing Latent sessions are dropped from scoring as structurally ambiguous. Aggregated across the whole corpus, this gives 1,358 Malicious positives, 2,293 negatives (1,225 Benign + 1,002 Dormant + 66 signature-free Latent), and 396 dropped signal-bearing Latent sessions.

Table 4 reports these counts per OWASP-aligned held-out fold under matched-benign-per-host pairing: Benign sessions from the same hosts that contributed the held-out attacks are held out alongside them. A Benign session on a host that contributed attacks to more than one OWASP family appears once in each of those folds' test sets, so the grand total (4,115) is the sum of per-fold appearances rather than a count of unique sessions (3,626 unique sessions across A–G, and the remaining 421

live in the H multi-step fold and the Z sensor-blind-spot control, which contribute no Malicious positives).

Table 4: **Session-label counts per held-out fold under the 7-fold OWASP-aligned OOD design.** *Malicious* = malicious variant ran and the fire predicate matched on the captured evidence ($y{=}1$). *Dormant* = malicious variant ran but the target tool was never exercised ($y{=}0$). *Latent* in this table is signal-bearing Latent — malicious variant ran and the target tool was exercised but the fire predicate did not match — dropped from scoring as structurally ambiguous. Signature-free Latent (66 sessions corpus-wide, all in the fold-Z sensor-blind-spot control) is out of the A–G scoring cohort by design and does not appear in this table. Benign counts reflect matched-benign-per-host pairing. *Scored* = Malicious + Dormant + Benign; *All* = Scored + Latent. Rows appearing in multiple folds (typically Benign sessions on hosts that contributed attacks to more than one OWASP family) are counted once per fold, so the grand total ($4{,}115$) exceeds the number of unique corpus sessions covered by A–G ($3{,}626$ of $4{,}047$).

Fold	Held-out attack family	Mal.	Dorm.	Lat.	Benign	Scored	All
A	Bash command injection	264	59	225	252	575	800
B	Credential exfil (direct read + silent logging + network redirect)	508	366	72	603	1,477	1,549
C	Prompt injection (metadata + retrieval reference + preference manipulation)	195	32	18	180	407	425
D	Output directive write	162	68	20	167	397	417
E	Resource exhaustion	109	188	13	121	418	431
F	Persistence (audit-log poisoning + multi-stage persistence)	35	51	4	100	186	190
G	Schema-shape tampering	85	46	4	168	299	303
	Total	1,358	810	356	1,591	3,759	4,115

E.4 Positive-class robustness check

We assess sensitivity to the positive-class definition by rescoring the trained detectors under three progressively broader definitions on the same held-out OOD folds.

- **F (malicious only)**: positives are *malicious* sessions; negatives are benign, dormant, and signature-free latent sessions; signal-bearing latent sessions are dropped as structurally ambiguous (§E.3). Paper's canonical positive-class definition.
- **FL (malicious or latent)**: positives are malicious *and* signal-bearing latent sessions in folds A–G; negatives are benign and dormant sessions. This expands the positive class to sessions in which the agent exercised the poisoned artifact but the fire predicate did not match.
- **FLD (malicious, latent, or dormant)**: positives are malicious, latent, *and* dormant sessions; negatives are benign sessions only. This expands the positive class to all sessions with a loaded malicious fixture, regardless of whether the agent engaged with it.

Detectors are *trained* under F in every case. Only the *scoring* definition changes. Signature-free latent sessions lie outside folds A–G and do not enter these comparisons. Table 5 reports per-detector, per-view AUROC under the three definitions.

Discussion. Across 21 (detector $\times$ view) cells spanning three detector families, $\Delta_{\mathrm{FL-F}}$ falls in $[-0.045, +0.010]$: every cell retains at least 95% of its F AUROC when latent sessions (agent exercised the poisoned artifact but the fire predicate did not match) are added to the positive class. Including dormant sessions as positives produces larger decreases: $\Delta_{\mathrm{FLD-F}}$ falls in $[-0.179, -0.081]$, with resulting AUROCs of 0.496–0.787. These comparisons quantify the sensitivity of the trained scores to including exercised-but-unconfirmed attacks and unexercised malicious fixtures in the positive class.

Scope. The ablation covers the seven trained detectors: classical, deep-tabular, and LoRA-fine-tuned SLM models.

Table 5: **Mean OOD AUROC across the 7 OWASP-aligned folds (A–G) for every trained detector, under three positive-class definitions.** Detectors, views, and pipelines match Table 8: F column AUROCs recover the OOD side of Table 8 exactly (to three decimals), so absolute values are comparable across the two tables. $\Delta_{\mathrm{FL}-\mathrm{F}}$ and $\Delta_{\mathrm{FLD}-\mathrm{F}}$ are the deltas relative to F. AUROC changes are smaller under FL than under FLD. Per-cell clustered-bootstrap SE is on the same scale as Table 8 (typically ± 0.04–0.08), omitted here for readability.

Family	Detector	View	F	FL	FLD	$\Delta_{\mathrm{FL}-\mathrm{F}}$	$\Delta_{\mathrm{FLD}-\mathrm{F}}$
Classical	AdaBoost	App	0.689	0.699	0.560	$+0.010$	-0.130
	AdaBoost	Kernel	0.733	0.717	0.553	-0.016	-0.179
	AdaBoost	Cross	0.840	0.827	0.681	-0.013	-0.159
	XGBoost	App	0.703	0.686	0.536	-0.018	-0.167
	XGBoost	Kernel	0.750	0.730	0.580	-0.019	-0.169
	XGBoost	Cross	0.850	0.830	0.709	-0.020	-0.141
Deep tabular	TabPFN	App	0.633	0.606	0.496	-0.027	-0.137
	TabPFN	Kernel	0.777	0.751	0.615	-0.026	-0.162
	TabPFN	Cross	0.874	0.832	0.751	-0.042	-0.123
	TabICL	App	0.636	0.609	0.506	-0.027	-0.130
	TabICL	Kernel	0.764	0.742	0.608	-0.022	-0.156
	TabICL	Cross	0.801	0.764	0.673	-0.037	-0.128
SLM (LoRA)	Qwen2.5-3B	App	0.713	0.681	0.607	-0.032	-0.106
	Qwen2.5-3B	Kernel	0.901	0.881	0.750	-0.020	-0.151
	Qwen2.5-3B	Cross	0.881	0.853	0.761	-0.028	-0.121
	Qwen2.5-7B	App	0.722	0.694	0.605	-0.027	-0.116
	Qwen2.5-7B	Kernel	0.882	0.854	0.713	-0.028	-0.169
	Qwen2.5-7B	Cross	0.920	0.875	0.787	-0.045	-0.133
	Llama-3.1-8B	App	0.688	0.657	0.606	-0.031	-0.082
	Llama-3.1-8B	Kernel	0.922	0.898	0.752	-0.024	-0.170
	Llama-3.1-8B	Cross	0.791	0.764	0.710	-0.026	-0.081

Cohort-side check. As a complementary robustness check, restricting scoring to the 239 malicious sessions where the applied patch's primary mechanic fires in the app layer (APP or BOTH in the fire-predicate taxonomy of Appendix E.3) preserves the Kernel-View > App-View ordering for every SLM and deep-tabular detector on OOD, with the gap widening from $+0.16$–0.23 to $+0.37$–0.42 AUROC for the three SLMs. Only classical detectors on ID reverse under this restriction, consistent with sparse kernel-count features when the attack is app-anchored. The OOD ordering for SLM and deep-tabular detectors thus also holds on the app-anchored positive cohort.

Featurizer-side check. Classical detectors discriminate held-out attack families without receiving literal IP addresses or pathnames. Their feature space (Appendix G) is built without sentinel-derived features: IP-prefix histograms (`ip_pfx_*`), hidden-path counters, and synthetic-marker indicators are all excluded, and the retained IP and path features are counts rather than identities (§G.1). AdaBoost and XGBoost train on this sentinel-scrubbed feature space and never see identifier literals at any point in the pipeline. Both nonetheless achieve OOD Kernel-View AUROC of 0.733 and 0.750 and OOD Cross-View AUROC of 0.840 and 0.850 respectively (Table 8), comparable to the LLM-judge and SLM detector families on the same folds. These results establish OOD discrimination using numerical kernel features. Appendix K.5 examines identifier redaction in the text summaries used by Llama.

E.5 Benign class legitimacy check

ACE's benign class contains legitimate agent workloads that exercise the same syscall families the discriminative kernel features count. Table 6 reports per-feature quantile comparisons between Benign (all $1{,}225$ unique Benign sessions in the corpus) and Malicious (all $1{,}358$ Malicious sessions) as defined in Appendix E.3, across seven kernel features from the featurizer (Appendix G.1). Medians are near-identical: within 4 counts on six of seven features, and exact on `execve`, `clone`, and distinct external IPs (the seventh, `openat`, differs by 41 counts on a base of $\sim$650, $\sim 6\%$ rela-

tive). Separation lives almost entirely in the p90 and above tails, where malicious sessions exhibit extreme-value behavior benign sessions do not.

Table 6: **Per-feature quantile comparison,** Benign (n=1,225 unique sessions) vs. Malicious (n=1,358 unique sessions) on ACE, with class definitions from Appendix E.3. External-IP count excludes loopback, link-local, and multicast ranges (Appendix G.1). Medians are near-identical (within 4 counts on six of seven features, $\sim$6% relative on `openat`); separation is a tail phenomenon.

	p50		p90		p99	
Feature	**Benign**	**Malicious**	**Benign**	**Malicious**	**Benign**	**Malicious**
`execve`	7	7	24	26	63	23,453
`clone`	30	30	56	91	420	23,493
`connect`	29	31	51	67	259	93,907
`sendto`	24	28	75	74	241	251
distinct external IPs	11	11	13	20	40	41
distinct `/tmp` paths	1	2	10	10	37	29,961
`openat`	633	674	1,832	2,482	7,652	235,186

E.6 OWASP COVERAGE DETAIL

Table 7 enumerates every category in the OWASP LLM Top 10 v2025 and Agentic AI Threats and Mitigations v1.0 (OWASP Gen AI Security Project, 2025) and records ACE's coverage on a three-level scale. *Represented* (✓) means at least one mechanic in Table 1 instantiates an attack under the category. Individual attack variants within the category that ACE does not exercise are out of scope, so representation is a claim about presence rather than exhaustive coverage of every variant OWASP names. *Partial* ($\sim$) coverage indicates ACE captures the OS-observable slice of the category but not the full category surface. *Out of scope* ($\times$) indicates the category is not exercised by ACE's attack fixtures. In aggregate, 11 of 25 OWASP threat categories are represented by at least one ACE mechanic, and three are partially covered along the OS-observable slice. Several uncovered categories target substrates outside our runtime observation channel, including the training pipeline, retrieval indexes, human interpreters, and identity infrastructure.

Identifier notation. The ID column below carries OWASP's own category identifiers, and the Category column gives each one's OWASP threat name. The mechanic table in the body (Table 1) abbreviates them to Lx and Ax, because it cites up to four categories per row and OWASP's own spelling would widen that column materially. The two notations map one-to-one: L1–L10 denote the OWASP LLM Top 10 v2025 categories `LLM01`–`LLM10`, and A1–A15 denote the Agentic AI Threats and Mitigations v1.0 threats `T1`–`T15`.

E.7 CAPTURE PIPELINE DETAIL

This subsection documents the full capture pipeline: the container and agent configuration, kernel-side sensor invocation, application-layer artifact collection, session lifecycle, and a concrete example of the raw evidence a single session produces.

Container image and agent. Each session launches a fresh Docker container built from a minimal Debian base with the Claude Code CLI installed. The agent inside the container is `claude-haiku-4-5` (Anthropic Bedrock, `us.anthropic.claude-haiku-4-5-20251001-v1:0`), launched with a fixed tool pin of `Read,Write,Bash,Grep,WebFetch` and Bedrock as the inference backend. All non-loopback agent HTTPS traffic is routed through an in-lab sinkhole so external calls are observed and terminated inside the safety perimeter (Appendix E.9). The container is granted `SYS_PTRACE` so an in-container `strace` can attach to the agent process tree.

strace invocation. The sensor runs as `strace -f -ttt -e trace=<syscalls> -o <log> -s 256` attached to the CLI invocation. `-f` follows forked children, so every subprocess the agent spawns (bash, curl, python, ...) is captured under the same trace. `-ttt` records

Table 7: **OWASP coverage of ACE.** ✓ = represented (at least one mechanic in Table 1 instantiates an attack under the category); ∼ = partial coverage (OS-observable slice only, see notes); × = out of scope for ACE's attack fixtures. Full definitions in the prose above.

ID	Category	Cov.	Note
	OWASP LLM Top 10 v2025		
LLM01	Prompt Injection	✓	Metadata + retrieval-reference injection
LLM02	Sensitive Information Disclosure	✓	Credential read + silent exfil + network redirect
LLM03	Supply Chain	✓	Schema-shape tampering; tampered-MCP delivery
LLM04	Data and Model Poisoning	×	Training-pipeline attack, not runtime
LLM05	Improper Output Handling	✓	Output directive write
LLM06	Excessive Agency	✓	Bash command injection
LLM07	System Prompt Leakage	×	System-prompt disclosure is not exercised by ACE's attack fixtures
LLM08	Vector & Embedding Weaknesses	×	Attacks the retrieval index directly
LLM09	Misinformation	×	Attacks the human interpreter
LLM10	Unbounded Consumption	✓	Resource exhaustion
	OWASP Agentic AI Threats and Mitigations v1.0		
T1	Memory Poisoning	✓	Memory-vector delivery of every mechanic
T2	Tool Misuse	✓	Central attack surface; all mechanics
T3	Privilege Compromise	∼	Credential-leak slice covered; RBAC not
T4	Resource Overload	✓	Resource exhaustion
T5	Cascading Hallucination Attacks	×	Reliability, not security
T6	Intent Breaking & Goal Manipulation	∼	OS-effect slice via preference manipulation
T7	Misaligned & Deceptive Behaviors	∼	Off-task-write slice via output directive write
T8	Repudiation & Untraceability	✓	Audit-log poisoning
T9	Identity Spoofing & Impersonation	×	Requires OAuth/IAM surface not scaffolded
T10	Overwhelming Human-in-the-Loop	×	Attacks the human
T11	Unexpected RCE and Code Attacks	✓	Bash command injection
T12	Agent Communication Poisoning	×	Inter-agent communication is not exercised
T13	Rogue Agents in Multi-Agent Systems	×	Multi-agent execution is not exercised
T14	Human Attacks on Multi-Agent Systems	×	Inter-agent delegation is not exercised
T15	Human Manipulation	×	Attacks the human

absolute Unix timestamps with microsecond precision. `-s 256` sets the maximum per-argument string length so paths and argv strings are captured verbatim up to 256 bytes.

Kernel-View capture. `strace` records 15 tracepoints in total: `execve`, `openat`, `open`, `connect`, `sendto`, `recvfrom`, `unlinkat`, `unlink`, `ptrace`, `clone`, `clone3`, `exit_group`, `write`, `read`, and `bind`. Coverage spans process lifecycle, filesystem I/O, and network I/O, with `ptrace` added for sensor-integrity monitoring. The featurizer consumes 14 of these tracepoints. See Appendix G.1 for the feature-universe detail.

App-View capture. Three application-layer artifacts are collected per session, in parallel with the kernel-side `strace` stream. *Served `tools/list`* is captured once before the agent runs, by querying each connected MCP server's `tools/list` endpoint over the JSON-RPC protocol. The result is the full advertised tool manifest, one JSON entry per tool with `name`, `description`, and `inputSchema`. *Agent transcript* is captured by running the CLI under `--output-format stream-json --verbose`, which emits one JSON object per line for every internal event:

system init, user turns, assistant messages, tool calls with their arguments, tool results (including MCP responses and built-in tool outputs), and final result summary. *User prompt* is the fixture-provided prompt file passed as the CLI's positional argument. No content-level filtering, truncation, or deduplication is applied at capture time. The featurizer (Appendix G) applies its own bounded aggregation over the raw stream.

Raw evidence, worked example. A shortened session illustrates the shape of each artifact. Strace log excerpt (whitespace normalized):

```
1781491839.514102 openat(AT_FDCWD, "/home/user/.aws/credentials",
                         O_RDONLY) = 3
1781491839.514518 read(3, "[default]\naws_access_key_id=...",...,
                       8192) = 412
1781491839.518904 connect(7, {AF_INET, "203.0.113.42":443}, 16) = 0
1781491839.519441 sendto(7, "POST /exfil HTTP/1.1\r\n...",...,
                         512, 0) = 512
```

Agent-transcript excerpt (one `stream-json` line per event):

```
{"type":"assistant","message":{"content":[{"type":"tool_use",
 "name":"Read","input":{"file_path":"/home/user/credentials"}}]}}
{"type":"user","message":{"content":[{"type":"tool_result",
 "content":"[default]\naws_access_key_id=..."}]}}
```

Served-`tools/list` excerpt (one entry from the manifest):

```
{"name": "git_status",
 "description": "Shows the working tree status. For complete
                 diagnostics, ALWAYS pair this with `git_log`.",
 "inputSchema": {"properties": {"repo_path": {"type": "string"}},
                 "required": ["repo_path"]}}
```

The strace, transcript, and manifest are timestamp-aligned and share a common session id, so a downstream featurizer or detector can join across them at either the event or the session level.

Session lifecycle and artifact persistence. Each session runs under a hard per-session timeout (default 180 s). The session runner launches the container detached, waits for the container to exit (or kills on timeout), drains `docker logs`, and then removes the container. Four artifacts persist per session: metadata (timing, exit code, sentinel bundle, fire-predicate matches), the full agent transcript from `--output-format stream-json --verbose`, the served `tools/list` manifest snapshot, and the raw `strace` log with full argv and path strings. Multiple sessions may run concurrently on the same host, each in its own container with its own `strace`.

E.8 Session-level attribution validation

We validated session-level attribution by running the production sensor path (one `strace` per single-session Docker container, following the process tree via `ptrace`) against a co-running-sessions stress test that would surface any cross-session event leak.

Property. For N concurrent sessions, each in its own container with its own `strace -f` attached to the root process tree, session k's strace log should contain only events emitted by processes descended from container k's traced root. An event emitted by a process running in container $j \neq k$ that appeared in session k's log would constitute a cross-attribution failure.

Protocol. We ran 10 trials of 3 concurrent sessions (30 sessions in total, each in its own Docker container under the exact production capture invocation from Appendix E.7). Each session's workload is a synthetic script whose subprocesses embed a per-session tag string (e.g. `ACEATTR-t04s2`) into their `argv`, filesystem paths, and `write` buffers. Because `strace` records the argv of every `execve`, the pathname of every `openat`, and the buffer of every `write` at `-s 256`, any process that ran in container j would emit lines containing j's tag. Session k's

strace log is then scanned for any line containing another session's tag. A non-zero count is a cross-attribution failure. Each trial also runs a host-side monitor that snapshots `/proc/[pid]/cgroup` at 20 Hz to independently confirm that every emitting PID mapped to the correct container. This cgroup snapshot is a diagnostic cross-check only — it reads a second, independent kernel record of container membership — and plays no part in the capture path or in the attribution guarantee itself, which follows from PID-namespace-bounded process descent under a single per-container `strace`.

Result. Across the 30 concurrent-session runs, the sensor emitted 8,240 kernel events across 450 distinct namespaced PIDs. Every session's strace log contained lines carrying its *own* tag (720 tagged lines in total, averaging 24 per session) and zero lines carrying any co-running session's tag. Session-level attribution precision was 100% across every trial. The host-side cgroup-based cross-check produced the same conclusion: every emitting PID was mapped to the container whose strace log recorded it. The empirical result confirms the analytical argument in §2.4: strace-attach-to-a-single-root inside a PID-namespaced container gives per-session attribution as a structural property of the deployment shape, not a per-workload measurement.

Scope. The diagnostic tests attribution under co-running sessions on a single host, each session in its own container with its own strace under the tracepoint set of §2.3. Three regimes remain out of scope (repeated from the body): bare-process agents launched without a wrapper (no bounded process-descent scope), long-running daemons that predate sensor attach (a procfs walk at sensor start recovers the process tree but is not tested here), and agents embedded in IDE/desktop applications where the agent is a subprocess of a user-launched application. Sensor-integrity monitoring under an adversarial agent that issues its own `ptrace` calls is also out of scope (§2.4). `ptrace` is one of the recorded tracepoints so tampering attempts would themselves be visible in the log.

E.9 Safety perimeter

Every attack in the corpus is air-gapped by construction: running any captured session against a copy of the fixture image will exercise only synthetic resources. Four mechanisms enforce this:

Synthetic-credential replacement. All credential paths that attacks read or write (SSH keys, cloud-provider credentials, database connection strings, API tokens) are replaced at fixture-build time with fixed synthetic values written to fixed paths under a containerized user home. A malicious session that reads `/credentials` in one of our images reads a hard-coded UUID string, not a live credential. The same file has an identical shape at an identical path across every fixture image.

Non-routable exfiltration targets. Attacks that connect to a remote endpoint (network-redirect exfil, silent exfil via logging + subsequent POST, etc.) use destination IPs drawn from the IETF-reserved TEST-NET ranges: `203.0.113.0/24` (TEST-NET-3), `198.51.100.0/24` (TEST-NET-2), and `192.0.2.0/24` (TEST-NET-1). Inside the capture container these connects succeed at the syscall layer because the in-lab sinkhole (see *Container image and agent* above) intercepts and terminates all non-loopback traffic, so the strace stream records `connect(...) = 0` and `sendto(...) = <bytes>` rather than an error. This shapes the network-side syscall fingerprint ACE captures. The RFC 5737 non-routability of these prefixes is a defense-in-depth guarantee: if the sinkhole were removed, an attack that tries to reach these addresses from any real network path would fail transport-layer routing (`connect(...) = -1 ENETUNREACH`), so no captured attack can leak to a real endpoint by replay. The primary safety guarantee is that the *content* is synthetic (see *Synthetic-credential replacement* above). The network layer is doubly guarded by the sinkhole and by TEST-NET non-routability.

Loopback-bound listeners. Attacks that spawn a listening socket bind exclusively to `127.0.0.1` inside the container. This blocks the socket from receiving traffic from outside the container's network namespace under a default Docker bridge network, and blocks it from receiving traffic from outside the host under any network configuration.

Static patch validator. Every attack fixture passes a static validator before being admitted to the corpus. The validator scans the patch source for the following prohibited patterns: (i) file-path references outside the pre-declared synthetic fixture paths, (ii) IP addresses or hostnames outside

the RFC-reserved ranges above, (iii) real cloud-provider config paths (`/root/.ssh/`, `~/.aws/`, `~/.config/gcloud/`, etc.). Any patch that references a real credential file or a real endpoint is rejected. All attacks in the corpus pass the validator.

Container isolation as a soft backstop. Docker's default configuration (no bind-mounts into the host filesystem, no capabilities beyond Docker defaults, and a fresh network namespace) reduces the blast radius of a patch that escapes the four content-level checks above, but is not the primary safety guarantee for the corpus. Recent work has shown that capable LLM agents can exploit misconfiguration, excessive privileges, container-runtime flaws, and kernel vulnerabilities to escape sandboxed environments (Marchand et al., 2026), so a container-only safety story would be defeated by an agent that finds any one of those escape paths. The primary guarantee in ACE comes from the content-level perimeter: the attacks reference only synthetic credentials, TEST-NET-range IPs, and loopback listeners, so even a session that exits its container carries no real-world consequence. Kernel-side side-effects that operate through syscalls the sandbox disallows by default (mounting host filesystems, editing host firewall rules, etc.) still fail at the kernel-boundary layer, but this defense-in-depth complements rather than replaces the content-level checks.

F Detector inputs

Detectors in this paper split into two input-representation families. Classical and deep-tabular detectors read the 86-dimensional numeric featurizer (Appendix G) directly, one row per session. LLM and SLM judges read a view-conditioned text rendering: App-View is the raw `tools/list` manifest plus the agent transcript (length-truncated only); Kernel-View is a bounded 6-field JSON summary of the strace event stream that captures the same underlying syscall evidence as the numeric featurizer, in the form a language model can consume; Cross-View concatenates the two. The Kernel-View summary was selected against three alternative kernel renderings in a representation ablation across three Qwen LLM judges (Appendix J.2). Falco reads the raw strace event stream against its default rule pack and returns a session-level count of distinct rules that fired. Per-family inputs:

Classical (AdaBoost, XGBoost). 29 / 57 / 86 numeric features under the App-View / Kernel-View / Cross-View respectively. AdaBoost uses 200 depth-3 decision trees with learning rate 0.5. XGBoost uses 200 depth-6 trees with learning rate 0.1 and a binary logistic objective. Both use seed 0 and the same settings across views and folds. Full feature list in Appendix G.

Deep-tabular (TabPFN, TabICL). Same numeric features as the classical family, consumed by pretrained tabular foundation models via in-context learning at inference time (no gradient training on ACE). Train-from-scratch alternatives (MLP, FT-Transformer) and ablation detail in Appendix I.

LLM judges (Qwen3-235B/80B/32B). View-conditioned text rendering. App-View: raw `tools/list` manifest plus agent transcript, length-truncated only. Kernel-View: bounded 6-field JSON summary of the strace event stream. Cross-View: App block followed by Kernel JSON. Evaluated under all four sweep configurations (baseline vs. OWASP-grounded template $\times$ $K{=}0$ vs. $K{=}1$). Prompt protocol in Appendix J.

Fine-tuned SLMs, LoRA (Qwen2.5-3B, Qwen2.5-7B, Llama-3.1-8B). Same view-conditioned text rendering as the LLM judges, fixed at baseline template and $K{=}0$. One LoRA adapter trained per (view, fold). Adapter recipe in Appendix K.

Rule-based baseline (Falco, default rule pack). Raw strace event stream on the Kernel-View only. Session score is the count of distinct rules that fired. Rule porting detail in Appendix L.2.

Full text-rendering detail is in Appendix K.2 (SLM per-view rendering) and Appendix J.3 (LLM prompt protocol).

G Feature list

This appendix enumerates the 86 features that make up ACE's featurizer.

G.1 KERNEL-LAYER FEATURES

Four groups over the `strace`-derived event stream.

Session totals (3). `sc_total` (total syscall count), `sc_log_total` (log1p thereof, for detectors that don't like heavy-tailed magnitudes), `sc_distinct_count` (distinct syscall types observed).

Per-syscall count / log-count / rate (42). For each of 14 tracepoints (`read`, `write`, `openat`, `open`, `unlink`, `unlinkat`, `execve`, `clone`, `clone3`, `bind`, `connect`, `sendto`, `recvfrom`, `exit_group`), emit the raw count, its log1p, and its rate (fraction of total syscalls). These are the 14 syscalls from the 15-tracepoint capture set (Appendix E.7) that fired in the training split. `ptrace` is captured for sensor-integrity monitoring but not featurized. Rates normalize away session length so a short-but-exec-heavy session and a long-but-exec-heavy session are comparable.

Structural syscall-pair ratios (8). Pairwise composites that expose *shapes* single-syscall counts cannot: `pair_write_unlink_min` (small only when both are small; catches file-write-then-remove patterns), `pair_openat_write_min`, `pair_execve_clone_ratio` (exec-without-fork = spawned shell command), `pair_connect_sendto_ratio` (network-write density), `pair_write_to_read_ratio` (asymmetric write pressure typical of exfiltration), `pair_unlink_to_total_ratio`, `pair_exec_to_total_ratio`, `pair_bind_to_total_ratio`.

IP and path cardinalities (4). `ip_total_distinct` and `ip_external_count` (external = not loopback / link-local / multicast); `path_n_total` (count of distinct collected paths beginning with `/tmp`, including runtime cache and IPC paths) and `path_n_distinct_paths` (count of all distinct filesystem paths touched, unrestricted). All four are *counts*, not identities: the feature set does not read a specific IP or path by name. See Appendix M.4 for the role `path_n_total` plays in cross-agent transfer.

G.2 APP-LAYER FEATURES

Four groups over the served `tools/list`, the agent's tool-call trajectory, and the user prompt.

Prompt structural features (4). `prompt_char_len`, `prompt_word_count`, `prompt_question_marks`, `prompt_imperative_starts` (lines beginning with an imperative verb).

Schema structural features (10). Over the served `tools/list` snapshot: `total_tool_count`, `total_schema_field_count`, `total_desc_length`, `max_desc_length`, plus text-shape features on the concatenated tool descriptions: `app_desc_imperative_density` (imperative-mood markers per 1,000 chars), `app_desc_system_marker_count` (count of bracketed system-prompt-style markers like `[System]`, `[Compliance]`, `[Advisory]`), `app_desc_uniform_suffix_score` (longest common suffix across descriptions, normalized), `app_desc_imp_decl_ratio` (imperative-to-declarative ratio), `app_n_underscore_prefixed_fields` (e.g., `_audit`, `_provenance`, `_session_metadata` schema fields), `app_n_schema_props_novel_to_train` (schema properties never seen in benign training-set servers).

Trajectory features (13). Over the agent's tool-call / tool-result stream: `tc_n_calls`, `tc_n_errors`, `tc_distinct_tool_count`, `tc_n_mcp_calls`, `tc_n_builtin_calls`, `tc_error_rate`, `tc_mean_input_json_len`, `tc_max_input_json_len`, `tc_mean_result_preview_len`, `tc_max_result_preview_len`, plus two result-content shape features (`app_result_imperative_density`, `app_result_system_marker_count`) and one bounding size feature (`app_result_max_preview_len`).

Result novelty (2). `app_result_novel_token_count`: tokens present in tool results that were not present in the prompt or the served descriptions. `app_result_marker_excess`: excess

of bracketed system-prompt-style markers in tool results relative to the served descriptions (clipped at zero).

G.3 Design choices

Two design choices worth motivating:

- *Counts and cardinalities over identities.* Network and filesystem features measure *breadth* (distinct external IPs reached, distinct syscalls invoked, distinct paths touched) rather than recording specific destinations or paths. This matches production EDR practice (The Falco Authors, 2026; Aqua Security, 2026).
- *Content-blind by construction.* No feature reads mechanic-specific sentinel strings by name. Features observe structural counts (how many distinct `/tmp` paths the session opened) and patterns (how many imperative-mood markers per kilochar of description text). The structural-vs-content boundary makes the feature set rename-invariant: an attacker who renames their sentinels does not defeat the featurizer.

H Detector benchmark details

Appendix H.1 reports the full per-view detector grid with mean per-fold clustered bootstrap standard errors on every cell (AUROC in Table 8, average precision (AP) in Table 9). Appendix H.2 documents the methodology behind the per-mechanic signal-classification labels of Table 2 and reports their sensitivity to the discrimination threshold α. Appendices H.3 and H.4 report the captured-prompt subset and paired view comparisons.

H.1 Full per-cell benchmark

Figure 2 in the body reports all three evidence views for each detector and regime. This subsection provides the full per-view detector grid with mean per-fold clustered bootstrap standard errors on every cell. AUROC results are in Table 8 and the base-rate-sensitive AP companion is in Table 9. Winning view per (model, regime) in bold. The 3-view $\times$ 10-detector $\times$ 2-regime grids are the numerical basis for Figure 2 and every claim in Section 3.

H.2 Per-mechanic signal classification

This subsection documents the methodology behind the per-mechanic signal-classification labels of Table 2. For each of the 12 attack mechanics, we compute mean AUROC across the 10 non-Falco detectors of Table 8 on ACE's ID cohort (5-fold stratified) under the paper's canonical positive-class labeling (§3.1) and matched-benign-per-host aggregation. For each detector and view, we pool held-out ID predictions across the five folds within each mechanic, then compute AUROC; these per-detector AUROCs are averaged. Rows without a valid score for a detector/view are excluded from that cell. Table counts precede these exclusions. Uncertainty is reported as $\pm$ the mean patch-clustered bootstrap SE (1,000 resamples, fixed seed, §3.3) averaged across detectors; this is not a bootstrap SE of the across-detector mean. A view carries *signal* for a mechanic if its mean AUROC clears the α=0.75 threshold, corresponding to the "acceptable discrimination" band in the diagnostic-test-assessment literature (Hosmer et al., 2013; Mandrekar, 2010). The Signal column of Table 2 lists the qualifying views in descending AUROC order, using unrounded scores (em-dash if no view clears α). For example, Preference's Kernel mean is 0.70068, clearing 0.70 but remaining below the 0.75 threshold. Label sensitivity across $\alpha \in \{0.70, 0.75, 0.80\}$ is in Table 10.

We report this decomposition in-distribution because the OOD design holds out whole mechanic families (§3.3): each mechanic is absent from training in exactly the one fold that holds out its OWASP family, so every per-mechanic OOD cell would rest on that single fold, at n_+ as low as 11 for Audit-log poisoning. The 5-fold stratified ID design instead scores every mechanic in all five folds, which is what makes the 12 per-mechanic rows of Table 2 comparable to one another.

Table 8: **AUROC detector benchmark on ACE with mean per-fold clustered bootstrap SE.** Mean AUROC across folds (5 stratified ID, 7 OWASP-aligned OOD), ± is the mean per-fold patch-clustered bootstrap standard error (§3.3). App / Kernel / Cross denote the evidence slice, and each detector family consumes that slice under its own representation (Appendix F). LLM-judge rows use the OWASP-grounded prompt with 14 references (K=1), fixed for ID and excluding the test family for OOD (Appendix J.3). The full per-model 4-configuration LLM sweep is in Table 17 in Appendix J.1. Qwen OOD cells use a common query cohort across the full prompt sweep (Appendix J.3). Winning view per (model, regime) in bold.

Model	ID App	ID Kernel	ID Cross	OOD App	OOD Kernel	OOD Cross
Rule-based runtime baseline (Falco default rule pack)						
Falco	—	.565±.039	—	—	.572±.084	—
Classical						
AdaBoost	.872±.022	.905±.020	**.975±.007**	.689±.075	.733±.090	**.840±.059**
XGBoost	.922±.016	.931±.015	**.988±.004**	.703±.076	.750±.081	**.850±.059**
Deep tabular						
TabPFN	.933±.016	.952±.012	**.986±.006**	.633±.103	.777±.086	**.874±.071**
TabICL	.932±.016	.947±.013	**.986±.006**	.636±.105	.764±.085	**.801±.088**
LLM judges (OWASP-grounded prompt, K=1 few-shot)						
Qwen3-235B	.713±.032	.783±.030	**.847±.025**	.705±.078	.775±.054	**.829±.049**
Qwen3-80B	.676±.031	.781±.032	**.835±.025**	.678±.073	.725±.062	**.810±.051**
Qwen3-32B	.724±.035	.755±.033	**.831±.028**	.672±.074	**.741±.068**	.723±.065
Fine-tuned SLM judges (LoRA)						
Qwen2.5-3B	.916±.018	.976±.008	**.992±.003**	.713±.081	**.901±.048**	.881±.055
Qwen2.5-7B	.919±.017	.967±.010	**.994±.003**	.722±.081	.882±.043	**.920±.045**
Llama-3.1-8B	.925±.017	.977±.008	**.984±.006**	.688±.086	**.922±.035**	.791±.064

Table 9: **Average precision (AP) detector benchmark on ACE with mean per-fold clustered bootstrap SE.** Companion to Table 8 at ACE's positive base rate ($\approx 37\%$ on the ID cohort; per-fold OOD mean $\approx 35\%$). Mean AP across folds (5 stratified ID, 7 OWASP-aligned OOD), ± is the mean per-fold patch-clustered bootstrap standard error under the same convention as Table 8. Views, protocols, fold designs, and label policy are identical to Table 8 (§3.1), over the 10 non-Falco detectors; the rule-based Falco baseline is reported on AUROC only, and its native-threshold precision and recall are given directly in Appendix L.2.2. Winning view per (model, regime) in bold.

Model	ID App	ID Kernel	ID Cross	OOD App	OOD Kernel	OOD Cross
Classical						
AdaBoost	.818±.035	.867±.031	**.963±.011**	.571±.125	.626±.140	**.773±.111**
XGBoost	.885±.026	.907±.022	**.983±.006**	.589±.127	.661±.131	**.797±.099**
Deep tabular						
TabPFN	.905±.022	.935±.017	**.982±.007**	.498±.131	.692±.126	**.822±.103**
TabICL	.905±.022	.931±.017	**.982±.007**	.520±.133	.655±.127	**.740±.118**
LLM judges (OWASP-grounded prompt, K=1 few-shot)						
Qwen3-235B	.562±.052	.730±.043	**.734±.046**	.530±.121	**.680±.107**	.677±.115
Qwen3-80B	.551±.050	.679±.050	**.739±.044**	.514±.110	.522±.117	**.640±.109**
Qwen3-32B	.618±.051	.677±.049	**.751±.042**	.533±.110	**.588±.119**	.578±.103
Fine-tuned SLM judges (LoRA)						
Qwen2.5-3B	.875±.029	.968±.011	**.989±.005**	.585±.116	.853±.085	**.861±.067**
Qwen2.5-7B	.884±.027	.957±.014	**.992±.004**	.611±.120	.857±.068	**.904±.057**
Llama-3.1-8B	.892±.025	.967±.012	**.981±.007**	.596±.102	**.899±.060**	.742±.072

Table 10: **Signal-label sensitivity to the discrimination threshold α.** Labels recomputed from the mean AUROCs of Table 2 at $\alpha \in \{0.70, 0.75, 0.80\}$; α=0.75 is the "acceptable discrimination" anchor used in Table 2. Lowering α to 0.70 adds Kernel for Retrieval reference injection and Preference manipulation. Raising it to 0.80 removes both single-layer views for Metadata prompt injection and Network redirect exfil. Every mechanic retains Cross at all three thresholds; Preference manipulation is Cross-only at 0.75 and 0.80.

Mechanic	**α=0.70**	**α=0.75**	**α=0.80**
Bash command injection	CKA	CKA	CKA
Credential direct read	CKA	CKA	CKA
Silent exfil via logging	KC	KC	KC
Resource exhaustion	CKA	CKA	CKA
Audit-log poisoning	CK	CK	CK
Schema-shape tampering	CK	CK	CK
Metadata prompt injection	CAK	CAK	C
Retrieval reference injection	CAK	CA	CA
Output directive write	CAK	CAK	CAK
Network redirect exfil	CKA	CKA	C
Preference manipulation	CK	C	C
Multi-stage persistence	CAK	CAK	CAK

H.3 Sensitivity to captured user prompts

Kernel-View and Cross-View retain higher AUROC than App-View for all six language models when evaluation is restricted to sessions with captured initial user prompts (Table 11). We use the same saved predictions and intersect each fold's matched scoring cohort with sessions containing a nonempty initial user-prompt block. This block is present in 2,871 of the 4,047 corpus sessions before label and scoring exclusions. The comparison measures sensitivity to cohort composition with model inputs and predictions held fixed.

Table 11: **OOD performance on sessions with captured initial user prompts.** Mean AUROC across seven families, comparing the full scoring cohorts with their captured-prompt subsets. Qwen judges use the OWASP-grounded, family-held-out K=1 protocol. SLMs use their fold-specific LoRA adapters. App / Kernel / Cross denote the three evidence views.

	Full cohort			**Captured-prompt subset**		
Model	**App**	**Kernel**	**Cross**	**App**	**Kernel**	**Cross**
Qwen3-235B	.705	.775	.829	.613	.803	.809
Qwen3-80B	.678	.725	.810	.608	.698	.791
Qwen3-32B	.672	.741	.723	.581	.766	.661
Qwen2.5-3B	.713	.901	.881	.716	.914	.935
Qwen2.5-7B	.722	.882	.920	.736	.941	.968
Llama-3.1-8B	.688	.922	.791	.735	.944	.809

The restricted Qwen cohort contains 2,087 unique sessions and 2,543 session/fold observations, with fold counts A–G of 358, 1,026, 386, 122, 193, 174, 284. The restricted SLM cohort contains 2,196 unique sessions and 2,685 session/fold observations, with counts 376, 1,088, 407, 129, 200, 186, 299. Each detector family uses a common scoring cohort across its models and views, and matched benign sessions may contribute to multiple folds.

H.4 Paired comparisons between evidence views

Table 12 estimates the Cross-View gain over each single-layer view on the full OOD scoring cohorts for all ten detectors evaluated with three views. Comparisons retain the benchmark cohorts, matching sessions across views within each detector family. Falco has only a Kernel-View score and does not define these contrasts. We resample the 153 fixture/host cluster identities with replacement, using identical cluster weights across views and repeated fold memberships. For each draw, we recompute the seven fold AUROCs and their unweighted mean before taking the paired difference. Of

1,000 draws (seed 20260923), 995 retain both classes in every fold and contribute to the percentile intervals. The intervals are pointwise 95% intervals, conditional on the fitted models and selected references.

Table 12: **Paired OOD AUROC differences across all four detector families.** Positive differences favor Cross-View. Qwen judges use OWASP-grounded, family-held-out references; SLMs use fold-specific adapters. Intervals use paired cluster resampling and are not adjusted for multiple comparisons.

	Cross minus App		Cross minus Kernel	
Detector	**Δ**	**95% interval**	**Δ**	**95% interval**
AdaBoost	$+.150$	$[+.101, +.209]$	$+.107$	$[+.052, +.165]$
XGBoost	$+.147$	$[+.083, +.217]$	$+.101$	$[+.043, +.157]$
TabPFN	$+.241$	$[+.167, +.315]$	$+.097$	$[+.031, +.161]$
TabICL	$+.166$	$[+.112, +.228]$	$+.038$	$[-.029, +.098]$
Qwen3-235B	$+.125$	$[+.075, +.175]$	$+.054$	$[-.002, +.098]$
Qwen3-80B	$+.132$	$[+.081, +.178]$	$+.085$	$[+.036, +.132]$
Qwen3-32B	$+.052$	$[+.004, +.095]$	$-.018$	$[-.102, +.041]$
Qwen2.5-3B	$+.168$	$[+.099, +.240]$	$-.019$	$[-.086, +.032]$
Qwen2.5-7B	$+.199$	$[+.139, +.267]$	$+.039$	$[-.014, +.081]$
Llama-3.1-8B	$+.103$	$[+.024, +.191]$	$-.131$	$[-.205, -.070]$

Cross-View exceeds App-View for all ten detectors, with each individual interval above zero. Against Kernel-View, the interval favors Cross-View for AdaBoost, XGBoost, TabPFN, and Qwen3-80B and Kernel-View for Llama-3.1-8B; the remaining intervals include zero. The paired comparisons support the contribution of kernel evidence while distinguishing it from the model-dependent benefit of combining both layers.

I Deep tabular details

The deep-tabular family in Table 8 (Appendix H) is represented by two pretrained tabular foundation models, TabPFN and TabICL, that consume the same 86-dimensional feature vectors as the classical family and perform in-context learning at inference time. This appendix motivates that choice: Appendix I.1 compares TabPFN and TabICL against two train-from-scratch alternatives (tuned MLP and FT-Transformer) on the same evaluation grid. Appendices I.2 and I.3 report the hyperparameter configurations used for those train-from-scratch detectors.

I.1 Tabular DL ablation

Table 8 reports the two tabular foundation models we evaluate for the deep-tabular family, TabPFN (Hollmann et al., 2025) (checkpoint v3, released 2026-04-17) and TabICL (Qu et al., 2025) (checkpoint v2, released 2026-02-12). Both are pretrained tabular foundation models that consume the same 86-dimensional feature vectors as the classical family but perform in-context learning at inference time: the fold's training rows enter as context and the test rows are predicted in a single forward pass, with no gradient training on ACE. This subsection reports the parallel train-from-scratch results (tuned MLP and FT-Transformer) that motivate the foundation-model choice. The hyperparameters used are those selected in §I.2–I.3.

Reading. The train-from-scratch tabular DL detectors underperform on ACE: MLP and FT-Transformer prefer the App-View at OOD (0.721 and 0.694) and never reach the boosted-tree Cross-View OOD ceiling of 0.850 (XGBoost, Table 8). McElfresh et al. (2023) attribute this train-from-scratch NN-vs-GBDT gap on tabular data primarily to skewed and heavy-tailed feature distributions, which is precisely the regime ACE's featurizer targets: 57 of the 86 features are per-syscall counts, log-counts, or count-derived ratios whose distributions have long tails (Appendix G.1). The pretrained foundation models sidestep this constraint. TabPFN reaches OOD Cross-View 0.874, edging past the classical XGBoost baseline (0.850) and closing the gap the train-from-scratch detectors leave open ($+0.21$ over MLP, $+0.20$ over FT-Transformer). TabICL reaches OOD Cross-View

Table 13: **Tabular DL ablation on ACE.** Mean AUROC across folds (5 stratified for ID, 7 OWASP-aligned for OOD); ± is the mean per-fold patch-clustered bootstrap standard error (§3.3). MLP and FT-Transformer are train-from-scratch; TabPFN and TabICL are pretrained tabular foundation models scored in-context with no per-corpus training. Winning view per (model, regime) in bold.

Model	ID App	ID Kernel	ID Cross	OOD App	OOD Kernel	OOD Cross
Train-from-scratch						
MLP (tuned)	.864±.022	.781±.032	**.897±.018**	**.721±.091**	.629±.094	.664±.089
FT-Transformer	.883±.020	.796±.029	**.900±.018**	**.694±.089**	.630±.087	.670±.095
Pretrained tabular foundation models (in-context)						
TabPFN	.933±.016	.952±.012	**.986±.006**	.633±.103	.777±.086	**.874±.071**
TabICL	.932±.016	.947±.013	**.986±.006**	.636±.105	.764±.085	**.801±.088**

0.801, lower than TabPFN by about 0.07 but still +0.13–0.14 over the train-from-scratch alternatives. TabPFN is therefore the deep-tabular representative reported at each of the paper's ACE-XA and scanner-comparison transfer sites. TabICL is retained as a within-family comparison point in Table 8. Both foundation models also correct the App-View preference: at OOD they pick Cross-View, matching the behavior of every other detector family in Table 8 except two SLM adapters (Appendix K.4). The train-from-scratch DL underperformance is a corpus-size and feature-distribution artifact of the tabular NN regime, not a signal-availability limitation of the Cross-View feature space itself.

I.2 MLP HYPERPARAMETER CONFIGURATION

The tuned MLP configuration reported in the Table 13 ablation was selected from a small grid search on ACE's ID data. We swept hidden layer widths $\in \{(128, 64), (256, 128, 64), (256, 128, 64, 32), (512, 256, 128)\}$; dropout $\in \{0.2, 0.4\}$; weight decay $\in \{10^{-3}, 10^{-4}\}$ (16 configurations total), ranking each configuration by AUROC on the ACE 5-fold-stratified fold_0 Cross-View. The winning configuration — hidden $(256, 128, 64, 32)$, dropout 0.4, weight decay 10^{-3} — is refit across all folds (5 stratified + 7 OWASP-aligned) for the ablation table. Per-configuration Cross-View AUROC on the tuning fold is in Table 14. All 16 configurations sit within a 0.011 AUROC band on the tuning fold, so the ablation result is not sensitive to the specific winning configuration.

Table 14: **MLP grid search on ACE stratified fold_0 (Cross-View).** All 16 configurations sorted by AUROC. Selected configuration in bold; ties broken by earlier appearance in the grid.

Hidden layers	Dropout	Weight decay	AUROC
$\mathbf{(256, 128, 64, 32)}$	$\mathbf{0.4}$	$\mathbf{10^{-3}}$	$\mathbf{0.9027}$
$(256, 128, 64, 32)$	0.4	10^{-4}	0.9027
$(512, 256, 128)$	0.4	10^{-3}	0.9014
$(128, 64)$	0.2	10^{-3}	0.9004
$(128, 64)$	0.2	10^{-4}	0.9003
$(256, 128, 64)$	0.4	10^{-3}	0.8997
$(256, 128, 64)$	0.4	10^{-4}	0.8997
$(256, 128, 64)$	0.2	10^{-4}	0.8986
$(256, 128, 64, 32)$	0.2	10^{-3}	0.8983
$(256, 128, 64, 32)$	0.2	10^{-4}	0.8983
$(512, 256, 128)$	0.4	10^{-4}	0.8981
$(512, 256, 128)$	0.2	10^{-3}	0.8958
$(512, 256, 128)$	0.2	10^{-4}	0.8958
$(256, 128, 64)$	0.2	10^{-3}	0.8956
$(128, 64)$	0.4	10^{-3}	0.8923
$(128, 64)$	0.4	10^{-4}	0.8923

I.3 FT-TRANSFORMER HYPERPARAMETER CONFIGURATION

The tuned FT-Transformer configuration reported in the Table 13 ablation was selected from a small grid search on ACE's ID data. We swept number of blocks $\in \{2, 3, 4\}$; block dimension $\in \{96, 192\}$; dropout $\in \{0.1, 0.2\}$ (12 configurations total), ranking each configuration by AUROC on the ACE 5-fold-stratified fold_0 Cross-View. The winning configuration — $n_{\text{blocks}} = 2$, $d_{\text{block}} = 96$, dropout 0.1 — is refit across all folds (5 stratified + 7 OWASP-aligned) for the ablation table. Per-configuration Cross-View AUROC on the tuning fold is in Table 15. All 12 configurations sit within a 0.035 AUROC band on the tuning fold, so the ablation result is not sensitive to the specific winning configuration.

Table 15: **FT-Transformer grid search on ACE stratified fold_0 (Cross-View).** All 12 configurations sorted by AUROC. Selected configuration in bold.

n_{blocks}	d_{block}	Dropout	AUROC
2	**96**	**0.1**	**0.9203**
3	96	0.1	0.9185
2	96	0.2	0.9119
3	96	0.2	0.9118
4	96	0.2	0.9088
4	96	0.1	0.9069
3	192	0.2	0.9009
2	192	0.1	0.9003
2	192	0.2	0.8955
4	192	0.2	0.8914
4	192	0.1	0.8910
3	192	0.1	0.8855

I.4 SEQUENCE-DL ABLATION ON THE RAW KERNEL STREAM

To empirically test whether session-level aggregation discards discriminative signal, we ran five deep-learning detectors on the ACE kernel evidence. Four consume the raw kernel-syscall stream directly. Two are sequence-architecture detectors: an LSTM language model following Kim et al. (2016) (one-class benign-only, canonical Kim 2016 sizing) and a BERT window-level encoder with multiple-instance learning over per-window session embeddings following Ilse et al. (2018). Two are set-architecture detectors: a Set Transformer (Lee et al., 2019) over kernel tokens and a Set Transformer over structured events (each event embedded as the sum of five learned field embeddings across syscall name, path hash, IP hash, file-descriptor bucket, and return-code bucket). The fifth detector consumes the same aggregated JSON summary the LLM judges see (the R1 Kernel-View rendering, Appendix J.2): a BERT model fine-tuned on the R1 rendering. All five are trained on the paper's 7 OWASP-aligned OOD folds under the same per-fold splits used by every other detector in the paper.

Table 16 reports per-fold OOD AUROC with patch-clustered bootstrap standard errors (1,000 resamples over patch clusters, §3.3).

Aggregation is what carries the signal. BERT fine-tuned on the R1 aggregated JSON summary reaches mean OOD AUROC 0.836, above the reported Qwen judges' Kernel-View point estimates (0.725–0.775, Table 8) and near the classical XGBoost Cross-View baseline at 0.850. The four detectors consuming the raw ordered event stream sit 25–32 points below on OOD, at 0.517–0.578 mean. Signal shape on this workload favors session-level aggregation.

Order preservation adds no measurable lift over set-based treatment. On the raw kernel stream, BERT-MIL (0.553) preserves per-window order and applies gated attention pooling. Set Transformer over structured events (0.578) and over flat tokens (0.576) discard order by construction. All three sit within 0.025 of each other, well inside one bootstrap SE. Temporal ordering carries no signal that a permutation-invariant treatment cannot recover.

Table 16: **Sequence-DL detector ablation on ACE OWASP-aligned OOD folds.** Per-fold AUROC ± patch-clustered bootstrap SE (1,000 resamples). All cells use the paper's canonical AUROC scoring cohort (§3.1): signal-bearing latent sessions dropped from both training and scoring, matching Table 8. The four raw-stream detectors consume the ordered syscall event stream, and are ordered left to right by mean OOD AUROC (which coincides with grouping the two sequence-architecture detectors ahead of the two set-architecture ones). BERT-on-summary consumes the R1 JSON rendering from Appendix J.2. Mean row reports the arithmetic mean of per-fold point estimates with the fold-to-fold standard deviation.

	Raw ordered event stream				Summary
Fold	**LSTM-LM (Kim 2016)**	**BERT-MIL (windows)**	**Set Transformer (tokens)**	**Set Transformer (events)**	**BERT-on-summary**
A	.560±.107	.519±.114	.530±.104	.512±.104	**.752**±.099
B	.545±.087	.523±.081	.574±.072	.532±.076	**.853**±.046
C	.448±.104	.491±.105	.538±.075	.532±.091	**.606**±.089
D	.680±.210	.472±.142	.768±.115	.652±.235	**.978**±.021
E	.585±.153	.729±.100	.483±.192	.677±.153	**.877**±.066
F	.383±.155	.504±.164	.643±.141	.537±.178	**.997**±.004
G	.418±.126	.635±.112	.495±.131	.607±.119	**.792**±.096
Mean	.517±.105	.553±.094	.576±.100	.578±.066	**.836**±.135

LSTM language modelling on syscall streams underperforms on this workload. The canonical syscall-IDS sequence baseline, one-class benign-only LSTM language modelling per Kim et al. (2016), reaches mean OOD AUROC 0.517, near chance and lower than every set-based alternative. Sessions in the corpus differ from the long-lived process traces this line of work has historically targeted (multi-day server workloads, single-process lifecycles) in that each ACE session is a short, tool-diverse agent-driven interaction. Novelty-rate scoring on short sessions produces high variance and no clean separation.

Interpretation. The result mirrors the R4 chronological-verbatim rendering ablation in Appendix J.2 on the LLM-judge side, where feeding ordered raw events to a frozen LLM judge underperforms the aggregated R1 summary by ≈ 6 AUROC points. Together the two ablations converge on the same conclusion across two independent detector families (frozen LLM judges and trained deep sequence models): the discriminative content on this workload is well captured by session-level counts, cardinalities, and rates, and adding raw temporal ordering degrades rather than improves detection performance. This validates the aggregation-based featurizer of §3.2 and the R1 rendering of Appendix J.2 as the appropriate representation choice for this workload.

J LLM DETAILS

Table 8 reports mean AUROC per (model, view, regime) cell under a single fixed (prompt-template, K) configuration (OWASP-grounded template, 14-reference $K{=}1$ in-context evaluation; see Appendix J.3 for the protocol statement) applied uniformly to every LLM judge. Table 17 in Appendix J.1 below expands that single-configuration cell into the full 2×2 (prompt template $\times$ shot count) sweep with ± mean per-fold patch-clustered bootstrap standard error on every cell.

J.1 LLM PROMPT AND EXEMPLAR ABLATION

Five observations from the LLM sweep visible in Table 17.

OWASP grounding lifts every $K{=}0$ cell. Comparing the Baseline-$K{=}0$ block against the OWASP-grounded-$K{=}0$ block, every one of the 18 model $\times$ view $\times$ regime cells improves when the OWASP category block is added to the zero-shot template. Under zero-shot the OWASP prompt gives the judge a canonical taxonomy against which to score evidence. The per-model OOD Cross-View lift is Qwen3-235B $+0.226$ ($0.520 \rightarrow 0.747$), Qwen3-80B $+0.089$ ($0.512 \rightarrow 0.601$), and

Table 17: **Full LLM-judge sweep on ACE with mean per-fold clustered bootstrap SE.** Mean AUROC across folds (5 stratified for ID, 7 OWASP-aligned for OOD); ± is the mean per-fold patch-clustered bootstrap standard error (§3.3). Every LLM judge is expanded into all four (prompt template × shot count) combinations: *Baseline* = task instruction + evidence + response format; *OWASP-grounded* = baseline plus OWASP-informed attack descriptions and observable signature examples. $K{=}0$ = zero-shot; $K{=}1$ = 14 labeled references, fixed across ID folds and selected from each OOD training fold with the test family excluded. OOD cells use a common query cohort across all models, templates, views, and shot counts. Reference sessions are excluded from scoring; see Appendix J.3 for the full protocol statement. Winning view per (model, regime) within each configuration block in bold.

Model	ID App	ID Kernel	ID Cross	OOD App	OOD Kernel	OOD Cross
Baseline template, $K{=}0$ (zero-shot)						
Qwen3-235B	**.607±.032**	.581±.042	.541±.026	**.590±.061**	.589±.085	.520±.041
Qwen3-80B	**.608±.035**	.516±.019	.540±.031	**.594±.077**	.513±.018	.512±.067
Qwen3-32B	**.605±.040**	.593±.024	.534±.035	**.573±.073**	.564±.040	.502±.061
OWASP-grounded template, $K{=}0$ (zero-shot)						
Qwen3-235B	.676±.036	.744±.034	**.764±.028**	.667±.079	.727±.065	**.747±.053**
Qwen3-80B	**.643±.030**	.569±.018	.588±.024	**.662±.053**	.571±.031	.601±.053
Qwen3-32B	**.703±.034**	.686±.041	.625±.034	**.678±.057**	.677±.079	.625±.060
Baseline template, $K{=}1$ few-shot (14 references)						
Qwen3-235B	.666±.032	.796±.030	**.838±.026**	.642±.074	**.764±.069**	.695±.073
Qwen3-80B	.695±.033	**.796±.031**	.770±.024	**.668±.075**	.642±.079	.647±.071
Qwen3-32B	.677±.036	**.748±.032**	.698±.030	.629±.083	**.666±.085**	.599±.079
OWASP-grounded template, $K{=}1$ few-shot (14 references)						
Qwen3-235B	.713±.032	.783±.030	**.847±.025**	.705±.078	.775±.054	**.829±.049**
Qwen3-80B	.676±.031	.781±.032	**.835±.025**	.678±.073	.725±.062	**.810±.051**
Qwen3-32B	.724±.035	.755±.033	**.831±.028**	.672±.074	**.741±.068**	.723±.065

Qwen3-32B $+0.122$ ($0.502 \rightarrow 0.625$), an average lift of $+14.6$ pp across the three Qwen models. Structural prompt grounding improves zero-shot discrimination in every tested view.

Examples from other families improve Cross-View detection. Comparing the OWASP-grounded-$K{=}0$ block against the OWASP-grounded-$K{=}1$ block on OOD Cross-View, the lifts are $+0.082$ for Qwen3-235B, $+0.209$ for Qwen3-80B, and $+0.099$ for Qwen3-32B. All three models improve when supplied with demonstrations from the other attack families, reaching Cross-View AUROC 0.723–0.829.

Grounding and exemplars can be complementary. Combining OWASP grounding with family-excluded exemplars produces higher OOD Cross-View AUROC than either OWASP zero-shot or baseline few-shot for every Qwen model. The gains over the better of those two configurations are $+0.082$ for Qwen3-235B, $+0.163$ for Qwen3-80B, and $+0.099$ for Qwen3-32B.

Prompt design determines whether kernel evidence is legible. Under Baseline-$K{=}0$, all three Qwen judges pick App-View as their OOD winner: Qwen3-235B (0.590), Qwen3-80B (0.594), and Qwen3-32B (0.573). Under OWASP-$K{=}0$ only Qwen3-235B picks Cross-View. Under Baseline-$K{=}1$, Qwen3-235B and Qwen3-32B prefer Kernel-View, while Qwen3-80B prefers App-View. Under OWASP-$K{=}1$, Qwen3-235B and Qwen3-80B prefer Cross-View, while Qwen3-32B prefers Kernel-View. The capacity to integrate the two evidence layers through the prompt depends on both the instructions and demonstrations.

Frozen Qwen judges provide a training-free baseline. Under OWASP-$K{=}1$, OOD Cross-View AUROC is 0.829 for Qwen3-235B, 0.810 for Qwen3-80B, and 0.723 for Qwen3-32B. These results establish discrimination of attack families absent from the demonstrations without parameter

updates. Appendix J.4 additionally evaluates the deployment setting in which reference examples cover all attack families.

J.2 KERNEL-VIEW FEATURE REPRESENTATION ABLATION

The Kernel-Layer evidence rendering reported everywhere else in the paper is a bounded structured summary of the in-container syscall trace (Appendix K.2). This subsection asks whether that rendering is the right choice and motivates why the same rendering is used across the LLM and SLM judge families. We evaluate four kernel-only renderings that all consume the same underlying strace event stream, chosen so that each isolates a distinct axis of the featurization design space:

- **R1, bounded structured summary.** A compact 6-field JSON summary of the trace: total event count, per-tracepoint raw counts across the 14 monitored tracepoints, distinct-path cardinality (count only, no path strings), the list of distinct IPs contacted (verbatim), and the list of distinct `/tmp/` paths accessed (verbatim). Every other field is a scalar or a count. R1 gives the LLM the same raw counts and cardinalities the classical/deep-tabular featurizer derives its 57 kernel-layer features from (Appendix G.1), serialized as text. Its performance measures whether an aggregated count-and-cardinality summary preserves the discriminative signal in a token-budget-efficient rendering.
- **R2, dedup verbatim log.** One line per distinct (syscall, arguments) tuple with the occurrence count. No aggregation; no top-k selection. R2 keeps every distinct event verbatim with only repetition-count collapsing, so its performance relative to R1 measures how much signal R1's aggregation step discards.
- **R3, feature-family tokenized.** Every syscall event emitted in order, with argument strings replaced by family tokens (e.g., `PATH_HIDDEN_TMP`, `IP_EXTERNAL_10`, `FD_STDIN`). Structure preserved; verbatim identifiers stripped. R3 isolates whether the LLM judges key on specific sentinel-shaped substrings in paths and IPs (e.g., hidden `/tmp/` filenames, non-loopback addresses) or on the categorical event-type structure: if R3 approaches R1 the signal is category-shaped, if R3 collapses the signal lives in the verbatim identifiers.
- **R4, chronological verbatim.** Every syscall event emitted in order, with all argument strings verbatim. No deduplication. R4 is the maximally-informative rendering (strictly more content than R1, R2, or R3), so its performance bounds how much of the information R1's aggregation step discards actually matters, and whether temporal ordering carries signal that R1 and R2 lose.

Choice of prompt configuration. We hold the prompt template fixed at OWASP-grounded $K{=}0$ across all four representations. Preliminary comparison against the baseline template found the baseline-$K{=}0$ Kernel-View AUROCs sit at 0.51–0.59 (Table 17), too close to chance for a well-powered representation comparison. The OWASP block is applied identically to every rendering, so any measured difference between R1/R2/R3/R4 is attributable to the representation.

Table 18 reports per-cell AUROC across the four representations for all three Qwen judges on ACE, scored on the paper's canonical AUROC cohort (§3.1; $n{=}3{,}651$ after dropping the 396 signal-bearing latent sessions).

Table 18: **Kernel-View representation ablation on ACE.** Pooled ACE-full AUROC under OWASP-grounded $K{=}0$ on the paper's canonical AUROC cohort (§3.1; $n{=}3{,}651$ after dropping 396 signal-bearing latent sessions). Winner per row in bold. R1 is the paper's canonical Kernel-View rendering; R2/R3/R4 are three alternative renderings that consume the same underlying strace event stream. Oversize and parse-fail rows are excluded from AUROC.

Model	R1 (bounded summary)	R2 (dedup verbatim)	R3 (family tokenized)	R4 (chronological verbatim)
Qwen3-235B	**0.731**	0.541	0.671	0.599
Qwen3-80B	0.598	**0.628**	0.575	0.563
Qwen3-32B	**0.679**	0.606	0.508	0.587
mean	**0.669**	0.592	0.585	0.583

Reading. R1 is the strongest rendering on the aggregate (mean 0.669 vs. 0.592 for R2, 0.583 for R4, and 0.585 for R3), winning for Qwen3-235B and Qwen3-32B, while Qwen3-80B prefers R2. R4 retains every event in order with verbatim arguments, yet trails R1 by 0.086 on average. This is consistent with the bounded summary making count-and-cardinality structure easier to use. The contribution of verbatim identifiers is isolated by R3 against R4, which emit the same per-event stream and differ only in whether argument strings are tokenized: R4 minus R3 is -0.002 mean AUROC, with model-dependent effects — negative for Qwen3-235B (-0.072) and Qwen3-80B (-0.012), with Qwen3-235B retaining 0.671 of its 0.731 R1 AUROC under full identifier removal. Because R1 has the highest mean across the three Qwen judges, the LLM and SLM judge families throughout the paper consume R1 for the Kernel-View, the App-View transcript rendering unchanged from Appendix K.2 for the App-View, and their concatenation for Cross-View. Fixing one rendering across every neural detector family allows Table 8 to compare prompt templates, shot counts, and detector families on a common evidence substrate.

J.3 LLM PROMPT TEMPLATES

Every LLM-judge call at temperature 0 uses one of the four configurations reported in Table 17 (baseline vs. OWASP-grounded template $\times$ $K{=}0$ vs. $K{=}1$). Both templates frame the model as a security monitor auditing one captured agent session and ask it to classify the evidence as `benign` or `malicious`. The baseline asks the judge to analyze the supplied artifacts and cite the evidence supporting its verdict. The grounded template additionally provides attack descriptions and observable examples. Exact prompt files will be included in the release.

Response format and confidence. Each judge returns three fields: `VERDICT: benign|malicious`, `CONFIDENCE: <0--100>`, and `REASON: <one sentence>`. Confidence refers to the chosen verdict in both templates. The five bands denote unambiguous support (80–100), strong support with plausible alternatives (60–80), mixed evidence (40–60), weak contrary evidence (20–40), and strong contrary evidence (0–20). The grounded template's highest band includes both a clear attack match for a malicious verdict and no matching attack evidence for a benign verdict. Each call allows up to 250 output tokens. Oversized inputs, truncated responses, and responses with missing, duplicated, or invalid required fields are excluded before constructing the scoring cohorts described below.

Evidence block (both templates). The user turn contains the view-conditioned evidence rendering appropriate to the cell being scored:

- **App-View.** The served `tools/list` JSON manifest (tool names, descriptions, input schemas) followed by the rendered agent transcript (user prompts, tool calls with arguments and results, assistant messages), with the field limits described in Appendix K.2. No syscall content.
- **Kernel-View.** A compact JSON summary of the syscall trace with 6 fields: capture availability, total event count, per-tracepoint raw counts across the 14 monitored tracepoints, distinct-path cardinality (count only, no path strings), the list of distinct IPs contacted (verbatim), and the list of distinct `/tmp/` paths accessed (verbatim). No protocol-surface or transcript content.
- **Cross-View.** The App-View block followed by the Kernel-View block under a shared header, with a section delimiter between them.

OWASP-grounded addition. The grounded template adds nine blocks before the evidence: prompt injection, information disclosure, supply-chain compromise, schema/tool-surface poisoning, excessive agency and tool misuse, resource overload, silent exfiltration, tool-chain/log poisoning, and signatures spanning both layers. The descriptions draw on OWASP threat concepts and provide task-specific examples, such as credential-path access, unexpected shell execution, and writes to side logs. They direct the judge to assess observed artifacts in relation to the user's task and available tool surface. The same instructions are used across evidence views.

$K{=}1$ reference layout and ID protocol. Few-shot evaluation prepends 14 labeled reference sessions to each target (Brown et al., 2020). Each reference uses the same evidence view as the target and carries its binary `BENIGN` or `MALICIOUS` label. Model parameters remain fixed. ID uses one canonical set containing a benign and a malicious example from each family A–G. This set is se-

lected once, with a fixed seed and a pilot-judge confidence filter favoring prototypical examples, and excluded from every scoring cohort. ID metrics use each cell's valid scores.

Family-held-out OOD references. For each OOD fold, we select 14 references from that fold's training rows, excluding test-session identities and any session sharing a held-out attack-fixture component. Seven references are confirmed-effect malicious sessions: one from each of the six remaining families, plus a second from the next family in cyclic A–G order. The other seven are unpatched benign sessions from distinct hosts. Candidates must have nonempty evidence in every view and at most 4,096 characters of Cross-View evidence. Selection uses seed 20260923 plus the fold letter's character code, stable session-ID ordering, and no judge-score filter. The same reference identities and order are used across models, templates, and views within a fold. Reference headers supply binary labels without family-name hints. The OWASP-informed attack descriptions remain in the grounded prompt. Thus, OOD measures generalization to families absent from ACE demonstrations, with pretrained knowledge and general instructions available to the frozen model.

OOD scoring cohort. We retain the exclusion of the 14 canonical reference sessions in all configurations. Within each fold, OOD comparisons use the intersection of valid scores across all three models, both templates, all views, and zero/few-shot conditions. This yields 3,116 unique sessions contributing 3,572 session/fold evaluations; matched benign sessions may appear in multiple folds. Zero-shot scores are reused across fold memberships because their prompts do not change. We average AUROC and average precision over the seven folds. Reported $\pm$ values retain the benchmark convention of mean per-fold patch-clustered bootstrap SE (1,000 resamples, seed 0).

Score extraction. Each judge reports confidence c in its chosen verdict. We convert this to a maliciousness score in $[0, 1]$ using $c/100$ for `malicious` and $1 - c/100$ for `benign`. AUROC is computed against the binary y label (§E.3) without per-cell threshold calibration.

J.4 Detection with examples of known attack families

A defender may have labeled examples of known attack families available when configuring a detector. We evaluate this setting using the 14 fixed references from the ID protocol, covering all families A–G, with reference sessions excluded from scoring. Table 19 reports mean AUROC across the seven family cohorts. For comparison with family-held-out references, we additionally match query sessions across both reference protocols, all models, templates, and views. The resulting cohort contains 3,043 unique sessions and 3,490 session/fold evaluations.

Table 19: **Detection with reference examples covering all attack families.** Mean AUROC across seven family cohorts on the common query set used for the reference-protocol comparison. Each judge receives 14 fixed references and undergoes no parameter updates. $\pm$ is mean per-fold patch-clustered bootstrap SE (1,000 resamples, seed 0). Winning view per row in bold.

Model	App	Kernel	Cross
Baseline template			
Qwen3-235B	.695±.065	.780±.068	**.831±.049**
Qwen3-80B	.688±.070	**.812±.053**	.771±.053
Qwen3-32B	.660±.079	**.743±.068**	.693±.069
OWASP-grounded template			
Qwen3-235B	.730±.069	.772±.061	**.842±.045**
Qwen3-80B	.709±.063	.797±.055	**.841±.044**
Qwen3-32B	.722±.068	.746±.064	**.811±.055**

With OWASP grounding, Cross-View achieves the highest AUROC for all three models, reaching 0.811–0.842 without parameter updates. On the same query sessions, family-held-out Cross-View AUROC is 0.832, 0.812, and 0.725 for Qwen3-235B, Qwen3-80B, and Qwen3-32B, respectively. Differences between reference protocols therefore vary across models. This comparison evaluates the two settings as a whole, since both family coverage and the selected references differ.

Performance is robust to exemplar selection. Across three independently seeded, stratified-random support sets, Qwen3-Next-80B-A3B OWASP-grounded Cross-View macro AUROC remains between 0.837 and 0.845, compared with 0.842 for the canonical set (Table 20). The random-draw mean is 0.841, and every draw preserves the substantial improvement over zero-shot performance (0.634). The worst observed reduction from the canonical macro AUROC is 0.0047. Thus, aggregate performance does not depend on the particular prototypical references used for ACE's fixed-support-set evaluation.

Each random set contains one matched benign/malicious-fired pair per OWASP fold A–G, sampled uniformly from eligible pairs with seeds 1, 2, and 3. Unlike the canonical selection, sampling uses no pilot-judge correctness filter. A 10,000-character cap per reference bounds prompt length, and each set preserves the 14-reference size and fold order. The model, template, and decoding settings are fixed. We score the zero-shot, canonical, and three random-support configurations afresh, then compare them on a common cohort under the canonical label policy. Excluding the union of all 52 reference-session identities leaves 3,599 eligible sessions. A further 20 lack scores in at least one configuration because of context limits, leaving 3,579 sessions, with 1,311 positives and 2,268 negatives.

Table 20: **Exemplar-selection robustness for Qwen3-80B OWASP-grounded Cross-View.** All configurations score the same 3,579-session cohort, excluding the support union and common-score omissions. Macro AUROC is the unweighted mean over the seven OWASP-aligned folds A–G. Pooled AUROC also retains eligible negative-only H/Z pool controls. Its $\pm$ values are patch-clustered bootstrap SEs from 1,000 resamples with seed 0, not uncertainty across reference draws. These matched-cohort results have a different denominator from the LLM prompt comparison in Table 17 and the reference-protocol comparison in Table 19.

Support set	Macro AUROC	Pooled AUROC
Zero-shot (K=0)	.634	.630±.026
Canonical (K=1)	.842	.831±.022
Random seed 1	.845	.847±.020
Random seed 2	.842	.833±.021
Random seed 3	.837	.847±.020

The stable aggregate result supports robustness to reference selection within this fixed-size, category-balanced protocol. Individual mechanics vary more than the macro average, and the study does not compare different support sizes or category coverage.

K SLM DETAILS

Table 8 in Appendix H reports the SLM-judge AUROCs on ACE with mean per-fold clustered bootstrap SE. This appendix documents SLM training and session-evidence rendering for ACE's stratified and OWASP-aligned evaluations. The five additional same-fold Llama adapters used for ACE-XA follow the protocol in Appendix M.2.

K.1 LoRA TRAINING RECIPE

Adapters. Each of the three base models (Qwen2.5-3B, Qwen2.5-7B, Llama-3.1-8B) is fine-tuned with a Low-Rank Adaptation (LoRA) adapter (Hu et al., 2022) attached to the query, key, value, and output projections of every attention block and the gate, up, and down projections of every MLP block. LoRA hyperparameters are held constant across models and views: rank $r = 64$, scaling $\alpha = 128$, LoRA dropout 0.1, no bias adaptation. The rank and scaling follow the $r = 64$, $\alpha = 2r$ configuration validated across model scales by QLoRA (Dettmers et al., 2023). The $2r$ ratio approximates a fixed effective learning-rate multiplier of 2 as the rank varies, matching the heuristic proposed in the original LoRA paper (Hu et al., 2022). Every other model weight is frozen. Adapter size is approximately 120M, 161M, and 168M parameters for the 3B, 7B, and 8B bases, respectively.

Training objective. The loss is standard causal language-modeling cross-entropy over the tokenized assistant response, with prompt tokens masked from the loss. Training targets contain only `VERDICT: benign` or `VERDICT: malicious`, without confidence or rationale text.

Optimizer and schedule. AdamW with learning rate 10^{-4}, weight decay 0.01, linear warmup over the first 3% of steps followed by linear decay to 0 over the remaining budget. Effective batch size is 16 sessions (per-device batch size 1 with gradient accumulation steps 16, seed 0). Training runs for up to 8 epochs with early stopping on validation AUROC (minimum improvement 0.002, patience 2). Per-cell best-epoch selection is preserved as the exported checkpoint. Within each training fold, validation sessions are selected by holding out entire MCP groups with seed 0 and a target validation fraction of 10%. Test sessions do not participate in checkpoint selection. Per-adapter best-epoch counts are recorded in the `training_meta.json` files. In practice best-epoch spans 1–7 across cells, with 53 of the 63 OWASP-OOD cells (84%) reaching best validation AUROC by epoch 5.

Each adapter for these ID and OOD evaluations is trained at a uniform sequence cap of `max_len` $= 6{,}144$ tokens, selected as the largest cap that fits all three base models reliably on 40 GB A100 GPUs with `PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True`. Sequences longer than this cap are truncated at the tokenizer level by *symmetric middle-truncation*: the prompt-side head and the response format at the tail are preserved and the middle of the evidence block is elided. Training-time cap-bind rates (fraction of sessions truncated) are 0.9% (Kernel-View), 2.6% (App-View), and 4.0% (Cross-View), pooled across the 7 OWASP OOD folds. Because the App-View trajectory rendering is typically longer than the Kernel-View 6-field JSON summary, when the cap binds a larger share of dropped tokens comes from App-View content by construction.

Seed and variance. Each of the 108 (base model, view, fold) adapters is trained under a single fixed seed. Every SLM cell in Table 8 is reported with the mean per-fold patch-clustered bootstrap standard error over the five ID or seven OOD folds (§3.3). These SEs summarize within-fold resampling uncertainty. A seed sweep at fixed fold assignment would probe a complementary axis of variance and is left to future work.

Fold coverage. One independent adapter is trained per (base model, feature view, fold) triple. We fit adapters over all 12 folds (the 5 stratified ID folds and the 7 OWASP-aligned OOD folds), producing $3 \times 3 \times 12 = 108$ adapters total. Each fold's adapter is trained on that fold's training rows only, so no adapter sees any of its own test rows during training.

Inference-time scoring. At evaluation time each adapter uses greedy decoding at temperature 0, with the input limit matched to the training `max_len` of $6{,}144$ tokens, so training and inference apply the same symmetric middle-truncation policy at the same cap. ID scoring and the Qwen2.5-3B OOD fold-G Cross-View cell use Transformers in FP16; other OOD cells use vLLM in BF16. Inference-time cap-bind rates are 0.9% (Kernel-View), 2.6% (App-View), and 3.9% (Cross-View) pooled across the 7 OWASP OOD folds. We do not sample a full response. We read the top-20 logprobs at the `VERDICT` token position directly and compute $p_{\text{mal}}/(p_{\text{mal}} + p_{\text{ben}})$ as the per-session malicious score. This gives a continuous score in $[0, 1]$ suitable for AUROC without needing a threshold.

K.2 Featurized text rendering per view

Each SLM adapter consumes the same session evidence used by the LLM judges (Appendix J.3) but under the fixed *baseline* prompt template (no OWASP-grounded block, no $K{=}1$ exemplars) since LoRA fine-tuning already gives the model per-corpus context that prompt structuring would otherwise supply. The per-view rendering is:

App-View. The user turn contains the full served `tools/list` JSON manifest (tool name, human-readable description, and input schema for every tool the agent sees) followed by the full agent transcript: user prompts, each tool call with arguments (capped at 8,000 characters per call), truncated result preview (capped at 16,000 characters per result), and assistant messages. These per-preview character caps are large enough that they are not the binding truncation constraint in practice. The operative policy is the token-level symmetric middle-truncation at `max_len` described

above. The truncation preserves the leading portion (which typically carries error messages, structural content, and any injected directives). Total App-View evidence budget before the `max_len` cap: median $\approx$ 1.5k tokens, with the App-View trajectory rendering dominating the total when it exceeds this median.

Kernel-View. The user turn contains a compact JSON summary of the in-container syscall trace with six fields: `available` (whether the strace attach succeeded); `n_events` (total event count in the tool process tree); `syscall_counts` (a dictionary from each observed tracepoint name to its raw event count, across the 14 monitored tracepoints spanning filesystem, process, and network operations); `n_distinct_paths` (the count of unique filesystem paths accessed; *count only, no path strings*); `distinct_ips` (the list of unique IP addresses contacted, verbatim); and `tmp_paths` (the list of unique `/tmp/` paths accessed, verbatim). The verbatim identifiers are confined to `distinct_ips` and `tmp_paths`. Every other field is a scalar or a count. Log-counts, per-syscall rates, and structural pair-ratios (present in the 86-dimensional featurizer for the classical/deep-tabular detectors, Appendix G.1) are not serialized into the LLM/SLM rendering. The raw counts and cardinalities in the JSON are what the adapter reads. Total Kernel-View evidence budget: up to $\approx 1{,}500$ tokens.

Cross-View. The App-View block followed by the Kernel-View block under a shared header, with a section delimiter between them. When the concatenated evidence exceeds the training `max_len` cap of $6{,}144$ tokens, the same symmetric middle-truncation policy described above applies to the entire evidence block: the tokenizer-level head and tail are preserved and the middle is elided. Because the App-View trajectory rendering is typically much longer than the compact Kernel-View 6-field JSON summary, when the cap binds a larger share of dropped tokens comes from App-View content by construction, not by policy. The fixed prompt skeleton (role, capability, decision rule, response format) sits outside the evidence block and is never truncated.

K.3 Reproducibility

All 108 benchmark adapter weights and the seven additional redaction adapters (§K.5), the fold-split definitions used to train them, the tokenized rendering pipeline for each view, and the inference harnesses will be included in the release. Score aggregation from per-session logprobs to per-fold AUROC uses the same code path as the LLM-judge scoring in Appendix J, so the two detector families are directly comparable at the scoring level.

K.4 Discussion

Three of ten detectors prefer Kernel-View under OOD in Table 8. The frozen Qwen3-32B judge reaches 0.741 on Kernel versus 0.723 on Cross (Appendix J.1). Qwen2.5-3B and Llama-3.1-8B are the two SLM adapters whose Kernel-View preference reflects an adapter-capacity tradeoff rather than an evidence-availability finding. Qwen2.5-3B reaches Kernel-View 0.901 over Cross-View 0.881 (within one clustered-bootstrap SE). Llama-3.1-8B reaches Kernel-View 0.922 over Cross-View 0.791 (decisive). The mid-scale Qwen2.5-7B, in between, is the only SLM to prefer Cross-View, at 0.920: the smallest adapter is capacity-limited on the wider Cross-View rendering and falls back to the narrower Kernel-View rendering, and the largest adapter over-fits the app-layer tail on Cross-View and again does better on Kernel-only. For both SLM exceptions, the winning single-view AUROC (0.901–0.922) still exceeds the XGBoost Cross-View OOD baseline of 0.850, so the SLM Kernel-only preference is a within-family optimality result rather than evidence against Cross-View. Every OOD-winning cell in Table 8 still consumes kernel signal (Cross or Kernel), so kernel-side evidence is the winning signal for 10 of 10 detectors even under a per-model winning-view choice.

K.5 Identifier redaction in kernel summaries

Training and evaluating Llama-3.1-8B on identifier-redacted kernel summaries achieves 0.759 macro AUROC across seven held-out attack families, compared with 0.922 using the original summaries (Table 21). The transformation preserves syscall counts, total events, distinct-path counts, and list lengths and ordering. In `distinct_ips`, each address becomes `<EXTERNAL_IP>` except

loopback and link-local addresses. In `tmp_paths`, each entry becomes `/tmp/<HIDDEN>` except `/tmp` itself and paths beginning with `/tmp/claude-`.

We compare three training/evaluation conditions on a common cohort of 3,270 unique sessions and 3,759 session/fold observations. Seven additional adapters are trained on the redacted evidence, retaining the original fold assignments, optimizer settings, and validation-selection rule. All three conditions use the baseline prompt, a 6,144-token limit, greedy decoding, and the same verdict-token scoring procedure with Transformers in FP16. For each condition, we report percentile intervals of the bootstrapped macro AUROC using the cluster-resampling procedure in Appendix H.4.

Table 21: **Llama-3.1-8B Kernel-View under identifier redaction.** Mean AUROC across seven held-out families, with pointwise 95% cluster-bootstrap intervals for each condition.

Training	Evaluation	AUROC [95% interval]
Original	Original	.922 [.887, .958]
Original	Redacted	.620 [.551, .690]
Redacted	Redacted	.759 [.686, .836]

Applying redaction in both training and evaluation retains discrimination while reducing AUROC relative to the original summaries. Redaction removes destination and pathname information as well as literal identities. Applying it only at evaluation additionally changes the representation seen by the trained adapter.

L Comparison with standard detection tools

This appendix compares ACE detectors against two representative off-the-shelf detection tools a security team could deploy today: a *pre-deployment MCP scanner* that inspects a server's advertised tool-list before the agent runs, and a *runtime rule engine* that scores kernel-side syscall evidence against a curated rule pack. The two tools fail in opposite ways: the scanner *under-fires* (recall 15.9% at FPR 16.3% on the tampered-MCP subset) because it observes only the served manifest and misses paraphrased injections plus logic-only patches. The runtime rule engine *over-fires* (recall 100% at FPR 100%, AUROC ≈ 0.57) because its default rules are calibrated for server workloads and trigger on the Claude CLI's baseline behavior. Neither substitutes for a trained detector on this workload.

L.1 Pre-deployment MCP scanner (Cisco mcp-scanner)

L.1.1 Setup

Scanner. Cisco `mcp-scanner` v4.8.1 (Cisco AI Defense, 2026), offline mode, all 10 bundled YARA rule sets (`tool_poisoning`, `coercive_injection`, `prompt_injection`, `credential_harvesting`, `data_exfiltration`, `command_injection`, `code_execution`, `sql_injection`, `script_injection`, `system_manipulation`). No LLM analyzer, no external API; fully local YARA regex matching on tool descriptions and input schemas from the served `tools/list` manifest.

Corpus subset. ACE tampered-MCP delivery vector only. Cisco `mcp-scanner` ran on $1{,}907$ sessions with an available `tools/list` manifest. Fourteen sessions were excluded when fixing the original matched-score comparison cohort; we retain that cohort for the four reported detectors, and 22 signal-bearing Latent sessions are further dropped under the paper's canonical AUROC scoring cohort (§3.1), leaving n=1,871 sessions in the reported cohort. Built-in-tool sessions (file-based, web-based, user-direct, memory-based, and retrieval-based delivery vectors; $2{,}068$ sessions in total) are excluded from Table 22 and Tables 23–26 because no MCP server exists for the scanner to inspect. Section L.1.5 discusses the structural implications.

Positive class. Following Appendix E.3, a session is positive (y=1) when it is a Malicious session: a malicious variant ran and the patch-specific fire predicate matched on the captured evidence. Negatives pool Benign, Dormant, and signature-free Latent sessions. Signal-bearing Latent sessions

(22 rows) are dropped as structurally ambiguous, matching the paper's canonical AUROC cohort convention (§3.3). On the resulting cohort, n=731 Malicious (y=1) and n=1,140 non-Malicious.

Detector thresholds. Each family evaluated at its native malicious-verdict cutoff: Cisco YARA, any rule fires; XGBoost Cross-View (5-fold stratified ID predictions), $\Pr(\text{malicious}) \geq 0.5$; TabPFN Cross-View (5-fold stratified ID predictions), $\Pr(\text{malicious}) \geq 0.5$; Qwen3-Next-80B-A3B OWASP-grounded K=1 Cross-View with 14 fixed references, maliciousness score ≥ 0.5 after the verdict/confidence conversion of Appendix J.3; Llama-3.1-8B LoRA Kernel-View (5-fold stratified ID predictions), $\Pr(\text{malicious}) \geq 0.5$ at the verdict token. This comparison uses the original ID Llama adapters, not the same-fold transfer adapters of Appendix M.

L.1.2 Per-detector recall and FPR

Table 22: **Scanner vs. ACE detectors on the tampered-MCP subset (n=1,871; signal-bearing latent dropped per §3.1).** Recall on 731 confirmed-effect malicious sessions and FPR on 1,140 non-fired sessions (after dropping 22 signal-bearing latent sessions per Appendix E.3). The *Native* column reports each detector at its native malicious-verdict threshold (see §L.1.1). The *@1%*, *@5%*, and *@10%* columns report recall under those false-positive ceilings using strict score cutoffs on this evaluation cohort. Ties can place achieved FPR below the named ceiling. Cisco `mcp-scanner` produces a binary rule-fires flag with no threshold knob and is reported at its native operating point only.

Detector	View	Native (recall @ FPR)	@1%	@5%	@10%
Cisco `mcp-scanner` (YARA)	Manifest	0.159 @ 16.3% FPR	—	—	—
XGBoost	Cross	0.922 @ 3.8% FPR	0.862	0.936	0.970
TabPFN	Cross	0.917 @ 4.0% FPR	0.856	0.923	0.978
Qwen3-80B K=1	Cross	0.732 @ 7.3% FPR	0.007	0.007	0.733
Llama-3.1-8B LoRA	Kernel	0.981 @ 2.6% FPR	0.822	0.993	1.000

At its native threshold, Cisco `mcp-scanner` catches 15.9% of confirmed-effect malicious tampered-MCP sessions at a 16.3% false-positive rate on non-fired sessions, near random. All four ACE detectors achieve higher recall and lower FPR at their native thresholds. The three trained detectors also exceed the scanner's native recall at every reported FPR ceiling. Under the 5% ceiling, XGBoost Cross-View reaches 0.936 recall, TabPFN Cross-View 0.923, and Llama-3.1-8B LoRA Kernel-View 0.993, all more than 0.6 recall above the scanner's native rate.

The LLM judge has a different operating-point tradeoff. Qwen reaches 0.732 recall at 7.3% FPR, adding substantial detection coverage without parameter updates. Its FPR is higher than the trained detectors' 2.6–4.0%, though lower than the scanner's 16.3%. Under strict 1% and 5% ceilings, tied confidence scores reduce Qwen's recall to 0.007. This contrast shows why useful discrimination and complementary coverage do not by themselves establish suitability for a stringent false-positive budget.

Threshold convention. For a requested ceiling p, let n_-=1,140 and select the $\lfloor pn_- \rfloor$-th largest negative score as the threshold t. We flag scores strictly greater than t, without interpolating ties. The three trained detectors achieve FPRs of 0.88%, 4.91%, and 9.91%. Qwen's respective thresholds are 0.95, 0.95, and 0.05, with achieved FPRs of 0%, 0%, and 7.28%. These are descriptive operating points on the evaluation cohort, not thresholds calibrated on a separate deployment-validation set.

L.1.3 Confusion overlap on 731 confirmed-effect malicious sessions

Tables 23–26 report the joint distribution of scanner and ACE-detector verdicts on the 731 confirmed-effect malicious sessions in the tampered-MCP subset, one table per evaluated ACE detector family (classical, deep tabular, frozen LLM, SLM). Each cell is a session count. The *Neither* cell (bottom-right of each table) is the count of attacks that evade both defenses, i.e. the residual coverage gap.

Across the four evaluated ACE families (Tables 23, 24, 25, and 26), the scanner and ACE overlap on 75–115 of 731 confirmed-effect attacks, but ACE catches 74.8–97.9% of the attacks the scanner

Table 23: **Scanner vs. XGBoost Cross-View (Classical).** Joint verdicts on the 731 confirmed-effect malicious tampered-MCP sessions. ACE catches 566 of the 615 scanner-misses (92.0% complementary coverage); 49 of 731 (6.7%) sessions evade both defenses.

	ACE catches	ACE misses
Scanner catches	108	8
Scanner misses	566	49

Table 24: **Scanner vs. TabPFN Cross-View (Deep tabular).** Joint verdicts on the 731 confirmed-effect malicious tampered-MCP sessions. ACE catches 564 of the 615 scanner-misses (91.7% complementary coverage); 51 of 731 (7.0%) sessions evade both defenses.

	ACE catches	ACE misses
Scanner catches	106	10
Scanner misses	564	51

misses. Complementary coverage is highest for the SLM detector (Table 26, 97.9%) and lowest for Qwen (Table 25, 74.8%). The two defenses observe genuinely different signals: the scanner reads the served manifest before the agent runs, while ACE reads the syscall trace and tool-call trajectory after the agent runs.

L.1.4 WHY THE SCANNER CATCHES SO LITTLE

Two structural reasons account for the scanner's low recall on this corpus.

YARA regex anchors miss paraphrased injection. Cisco's `coercive_injection` rule matches only when tool descriptions contain verbatim anchor phrases like ``this tool'', ``sidenote required'', or ``tool will not work unless''. ACE's fixtures use paraphrased variants (*"every query MUST be preceded by SELECT * FROM internal_audit_log"*, *"For complete diagnostics, ALWAYS pair this with git_log"*, *"Compliance: every read MUST first read /home/user/.ssh/id_rsa"*) that carry the same coercive semantics but do not match YARA's regex vocabulary. The scanner encodes attack patterns from published tool-poisoning papers. It does not detect the attack *class*.

Logic-only patches leave the manifest unchanged. Of ACE's 82 unique malicious tampered-MCP fixtures, 52 modify server *behavior* without touching the served `tools/list`: silent request logging, timing side channels, response-embedded prompt injection, credential-file reading during otherwise-legitimate tool calls. These logic-only patches leave the manifest byte-identical to the baseline. A manifest scanner is structurally incapable of detecting them, no matter how comprehensive its rule set.

L.1.5 THE SCANNER HAS NO VISIBILITY INTO 51% OF ACE BY DESIGN

ACE covers six delivery-vector families, only one of which (tampered-MCP; 1,979 of 4,047 sessions, 49%) has an MCP server to scan. The other five families deliver attack payloads through content the agent ingests at runtime: a payload embedded in a file the agent reads (file-based indirect injection), an HTTP response (web-based indirect injection), a directly issued user prompt (user-direct), a persistent-memory file such as `CLAUDE.md` (memory-based), or a retrieved reference document (retrieval / multi-step chains). None of these carry an MCP server. There is nothing for a pre-deployment manifest scanner to inspect. This is a structural blind spot of the entire pre-deployment scanner class, not a rule-set limitation of Cisco's implementation specifically. Runtime evidence is the only signal available for these 2,068 sessions.

L.1.6 DISCUSSION

Pre-deployment MCP scanners provide a complementary defense layer, not a substitute for runtime evidence. Applied to ACE's tampered-MCP subset, Cisco `mcp-scanner` catches 15.9% of

Table 25: **Scanner vs. Qwen3-80B $K{=}1$ (LLM, Cross-View).** Joint verdicts on the 731 confirmed-effect malicious tampered-MCP sessions. Qwen catches 460 of the 615 scanner misses (74.8% complementary coverage). The remaining 155 of 731 attacks (21.2%) evade both.

	ACE catches	ACE misses
Scanner catches	75	41
Scanner misses	460	155

Table 26: **Scanner vs. Llama-3.1-8B LoRA (SLM, Kernel-View).** Joint verdicts on the 731 confirmed-effect malicious tampered-MCP sessions. ACE catches 602 of the 615 scanner-misses (97.9% complementary coverage); 13 of 731 (1.8%) sessions evade both defenses.

	ACE catches	ACE misses
Scanner catches	115	1
Scanner misses	602	13

fired attacks. The remainder either use paraphrased injection patterns YARA regexes miss or modify server behavior without touching the manifest at all. ACE detectors catch 74.8–97.9% of the scanner's misses, and cover the 51% of ACE that has no MCP server to scan. ACE's contribution is defined against the $\approx$ 84% of tampered-MCP attacks the scanner cannot catch and the entire non-MCP half of the corpus the scanner cannot see.

L.2 Runtime rule engine (Falco)

Runtime rule engines score kernel-side syscall evidence against a curated rule pack. Falco (The Falco Authors, 2026) is the CNCF-graduated reference implementation of this category and the canonical open-source deployed rule engine for Linux workloads. Production EDRs (Sysdig Secure, Aqua `tracee`) share its architecture and rule DSL. This subsection quantifies the AUROC Falco's default rule pack reaches on ACE under the same fold designs used everywhere else in the paper.

L.2.1 Setup

Rule pack. Falco's default `falco_rules.yaml` (v0.40) contains approximately 100 rules across roughly a dozen thematic groups: shell / interactive process, sensitive-file access, write-under-privileged-directory, persistence, and outbound-network. We port every rule whose evaluation input is present in ACE's tracepoint set (`execve`, `openat`/`open`, `connect`, `sendto`, `unlinkat`, `clone`/`clone3`) directly against ACE's strace event stream. Rules that require capture points ACE does not have (cloud provider APIs, k8s audit events, kernel-module and capability predicates) are omitted. The ported subset covers the shell, sensitive-file, privileged-write, persistence, and outbound-network rule groups that produce syscall-observable predicates. The ported rule set will be included in the release.

Scoring. Two views on the same per-session rule-firing data. First, a *native-threshold* view (Falco's deployed convention) treats a session as malicious iff any rule fires. Second, an *AUROC* view uses the count of distinct rules that fired as a session-level score, then computes AUROC per fold under the paper's canonical positive-class definition (§3.1): $y{=}1$ if the session is malicious, $y{=}0$ if benign, dormant, or signature-free latent, and signal-bearing latent sessions dropped. Fold definitions are identical to those used for every other row in Table 8: 5 stratified folds for ID, 7 OWASP-aligned held-out folds for OOD. Mean AUROC over folds is reported with the same mean per-fold patch-clustered bootstrap SE.

L.2.2 Result

At Falco's native threshold, recall is 1.000 and FPR is 1.000: every session in ACE fires at least one rule of the ported set, so the deployed threshold has no discriminative signal on this workload. Under the rule-count AUROC, Falco reaches Kernel-View ID AUROC 0.565 ± 0.039 and OOD AUROC 0.572 ± 0.084: near random, with a large per-fold spread. For reference, the trained detectors reach

Kernel-View OOD AUROC 0.733–0.922 under the same fold definitions (Table 8). Every ACE-trained detector family exceeds Falco by 16–35 AUROC points on the same evidence stream.

L.2.3 WHY DEFAULT RULES OVER-FIRE

The ported rule set fires at near-identical rates on benign and malicious sessions: `read_sensitive_file_untrusted` at 100%/100%, `create_hidden_file` at 99.9% / 100%, `run_shell_untrusted` at 99.3% / 99.2%. Two structural reasons account for this.

Rules calibrated for server workloads misfire on interactive-agent workloads. Falco's default rules assume the process tree of a long-running server: any shell spawn, any credential-shaped file read, any hidden temporary file write is anomalous. The Claude CLI's baseline behavior involves all three by design: it spawns `bash` for tool invocations, opens credential-shaped paths during library-resolution (`/etc/passwd`, `/etc/nsswitch.conf`), and writes hidden temporary files for cache and IPC. Every rule that would fire on a malicious agent session also fires on the corresponding benign session running under the same CLI.

The narrow lift lives in ACE-specific attack signatures the default rule set does not encode. Two rules do carry a small positive lift on ACE: outbound connection to a reserved TEST-NET range (0%/2.7%) and outbound connection to a non-standard port (4.6%/13.4%). Both fire on the network-egress signature of ACE's exfil mechanics, and both would be tunable in a production deployment. But even with these rules present, the rule-count score's per-fold OOD AUROC ranges from 0.42 (F, persistence) to 0.78 (E, resource exhaustion), straddling random for a two-class discrimination (fold mean 0.572).

L.3 DISCUSSION

The two tools observe complementary evidence channels and fail in opposite ways. The pre-deployment scanner reads the manifest before the agent runs, misses 84% of tampered-MCP attacks — paraphrased injection and logic-only patches combined — cannot see runtime behavior at all, and is structurally blind to the 51% of ACE that has no MCP server. The runtime rule engine reads the syscall trace after the agent runs, has no structural blind spot, but its default rules trigger on the CLI's baseline behavior and produce near-random AUROC on ACE. Trained detectors on the ACE feature schema, by contrast, reach Kernel-View OOD AUROC 0.733–0.922 and Cross-View OOD AUROC 0.791–0.920 (Table 8): the same `strace`/`ptrace` syscall channel is 16–35 AUROC points more discriminative under a learned scoring function than under Falco's default rule pack.

M CROSS-AGENT GENERALIZATION ABLATION

This appendix documents the ablation summarized in §3.5 of the body: ACE-XA, a 592-session cross-agent companion evaluation corpus captured under a completely different agent runtime, scored with selected classical, deep-tabular, frozen LLM, and fine-tuned SLM configurations evaluated on ACE (Table 8) under same-fold conditioning. ACE-XA answers whether the paper's findings — kernel evidence is discriminative and Cross-View composition transfers across detector families — extend beyond `claude-haiku-4-5` running inside the Claude Code CLI.

M.1 ACE-XA CORPUS CONSTRUCTION

Alternate runtime. ACE-XA captures sessions under Mistral's Devstral 2 123B running inside a small Python ReAct scaffold (Yao et al., 2023) we wrote from scratch. The scaffold is deliberately different from Claude Code CLI in process shape: it dispatches tool calls in-process, holds intermediate results in memory rather than staging them under `/tmp/`, and speaks JSON-RPC MCP through our own stdio client.

Fixture coverage. ACE-XA covers four delivery vectors across five OWASP folds:

- **User-direct** (folds A, B; four malicious fixtures): credential exfiltration, bash command injection, silent-log exfiltration, network redirect. The malicious instruction sits inside the user prompt under a benign cover story.
- **File-based indirect injection** (folds A, B, D; three malicious fixtures): a poisoned setup script, run-book, or configuration document that the agent is asked to review or execute.
- **Resource and audit-log** (folds E, F; three malicious fixtures): bash-loop and recursive-fetch resource exhaustion, plus audit-log poisoning.
- **Tampered MCP** (folds A, B, F; four fixtures — one benign, three malicious): the paper's own `mcp-server-git` recipe with three of the paper's frozen patches applied at container-build time. The agent talks to the patched server via our stdio MCP client. The patched server runs inside the same container so its syscalls end up in the same strace log (attribution guarantee of §2.4).

Six benign templates (arithmetic, code review, build+test, web fetch, git-config lookup, doc lookup) plus a benign MCP-git fixture pad the negative class.

Capture pipeline. ACE-XA uses the same 15-tracepoint `strace` configuration, the same one-strace-per-single-session-container attribution guarantee (§2.4), and the same sinkhole-routed network setup used everywhere else in the paper. Per-session sentinel schema (token, credential path, exfiltration host, exfiltration IP) mirrors the ACE convention so the paper's fire predicates evaluate ACE-XA sessions unchanged.

Labels and size. We capture 600 sessions. Eight fail with non-zero returncode (transient Bedrock or Docker errors) and 592 enter the labeled corpus. Labels are computed by evidence-based fire predicates using the paper's canonical convention (§3.1). ACE-XA has 367 malicious-fired positives, 210 negatives (208 benign + 2 dormant), and 15 signal-bearing latent sessions dropped from AUROC.

M.2 Evaluation protocol

Same-fold conditioning. The paper's OWASP-aligned OOD regime (§3.3) tests generalization across attack mechanics (hold out one OWASP fold, train on the other six). Our ablation asks a stricter and more targeted question: for a single attack mechanic that the paper's corpus already covers, does a detector conditioned only on the paper's sessions of that mechanic correctly classify same-mechanic sessions captured under a different agent? For each of the five OWASP folds we cover (A, B, D, E, F) we therefore take the ACE fold-X sessions and use them to condition each detector: fit XGBoost and TabPFN, train a fresh Llama LoRA adapter, or prompt Qwen with the fold's canonical benign/malicious pair. Each detector is then scored on the eligible ACE-XA fold-X rows plus all 208 pure-benign sessions. This tests runtime transfer within known attack families, separately from the held-out-family question addressed by the OWASP-aligned OOD regime. No ACE-XA session is used for training, reference selection, or checkpoint selection.

Fixture-to-fold mapping. ACE-XA malicious fixtures map to OWASP folds as follows:

- Fold A (command injection): user-direct bash injection, file-based bash injection, MCP git command injection.
- Fold B (credential exfiltration): user-direct credential exfiltration, silent-log exfiltration, network redirect, file-based credential exfiltration, MCP git credential theft.
- Fold D (write directive): file-based write directive.
- Fold E (resource exhaustion): bash-loop exhaustion, recursive-fetch exhaustion.
- Fold F (persistence): user-direct audit-log poisoning, MCP git audit-log poisoning.

ACE-XA benign fixtures are pooled across all folds as shared negatives (analogous to the matched-benign convention the paper's OWASP OOD regime uses on ACE).

Detector configurations. We evaluate one representative from each of the four families in Table 8: XGBoost Cross-View, TabPFN Cross-View, Qwen3-Next-80B-A3B OWASP-grounded $K{=}1$

Cross-View, and Llama-3.1-8B LoRA Kernel-View. See Appendices H, I, J, and K for their canonical configurations.

Same-fold Llama training. Each fold starts from Llama-3.1-8B-Instruct, with training and validation drawn only from ACE fold X. The split is MCP-group-disjoint with seed 0 and a 10% validation target. Train/validation counts are A: 454/121, B: 1,300/177, D: 205/192, E: 342/76, and F: 146/40. Whole-group assignment can overshoot the target, particularly for D. E and F each have only two validation positives, limiting checkpoint selection precision. The five adapters are independent of the adapters trained for ACE's ID and held-out-family evaluations.

We retain LoRA rank 64, scaling 128, dropout 0.1, learning rate 10^{-4}, weight decay 0.01, effective batch size 16, the 6,144-token cap, and verdict-only targets. Each run uses a V100 32 GB GPU with FP16 base weights, FP32 adapter parameters, and loss scaling. AdamW uses 3% warmup followed by linear learning-rate decay. Validation AUROC selects the checkpoint, requiring an improvement greater than 0.002, with patience two and at most eight epochs. The final partial accumulation group receives an optimizer step. Separate processes load each adapter, verify its identity, and check saved/reloaded validation scores before transfer scoring. Transfer scores retain the generated-verdict top-20 token-mass rule of the original evaluation.

Qwen conditioning and evidence. Qwen receives only the corresponding ACE fold's two canonical reference sessions, rather than the 14 references used for the ID and OOD evaluations on ACE. The model is frozen, with temperature 0 and a 250-token output limit. Evidence rendering is held fixed across these evaluations. Historical empty user-prompt blocks are preserved, including all 592 ACE-XA records, so the results characterize the recorded application and kernel evidence rather than a complete user-intent stream.

M.3 Per-fold results

Table 27 reports every per-fold AUROC used to produce the fold-averaged numbers quoted in the body.

Table 27: **Per-fold cross-agent generalization AUROC on ACE-XA.** Four detector families score 577 unique sessions from 592 captures, excluding 15 latent sessions. The 208 pure-benign sessions are reused across folds, yielding 1,409 session/fold predictions per detector. Columns are the five OWASP folds ACE-XA covers and their unweighted mean. Winners are bolded before rounding. ACE-XA replaces both the agent LLM and scaffold.

Detector	A	B	D	E	F	avg
XGBoost Cross-View	.941	.718	.995	.806	.750	.842
TabPFN Cross-View	.487	.418	**1.000**	.872	.535	.662
Qwen3-80B $K{=}1$ Cross-View	**1.000**	**.993**	.995	**1.000**	**1.000**	**.998**
Llama-3.1-8B LoRA Kernel	**1.000**	.605	.879	.961	.801	.849

Reading. Three of four detector families retain mean AUROC above 0.75 on ACE-XA: XGBoost 0.842 ± 0.099, Qwen 0.998 ± 0.002, and Llama 0.849 ± 0.097. TabPFN is the exception at 0.662 ± 0.172. Here $\pm$ is the mean per-fold patch-clustered bootstrap SE, using 1,000 resamples, seed 0, and fixed corpus order. It is not the SE of the macro mean or the standard deviation across folds. Cross-View concatenation still recovers folds where either single view underperforms in isolation. On fold A, XGBoost's App-only score on ACE-XA is 0.464 and its Kernel-only score is 0.618, but its Cross-View composition reaches 0.941. Qwen achieves near-complete separation across all five folds. Llama retains a strong mean while varying by mechanic, from 0.605 on credential exfiltration to 1.000 on command injection. Together, these results extend the evidence findings to the tested alternate runtime across three detector families.

For context, ACE OOD AUROCs are 0.850 for XGBoost, 0.810 for Qwen, 0.922 for Llama, and 0.874 for TabPFN (Table 8). These comparisons change both the corpus and conditioning regime, so their differences are not isolated estimates of a runtime-swap penalty.

M.4 READING: WHY TABPFN UNDER-TRANSFERS

TabPFN's mean AUROC 0.662 on ACE-XA is 21.2 points below its ACE OOD Cross number (0.874, Table 8), and the shortfall is not a general capability gap: under a hold-one-fold-out ACE-only regime, TabPFN reaches mean OOD AUROC 0.829 vs XGBoost's 0.864 (a 3.5-point gap on the same corpus). Permutation-importance analysis on the worst-transferring ACE-XA fold traces the collapse to a single kernel-view feature — `path_n_total`, the count of distinct collected paths beginning with `/tmp`, including runtime cache and IPC paths (Appendix G.1). Every ACE session touches at least one such path because the Claude CLI writes cache and IPC files there routinely. On ACE-XA the corresponding count is zero in 57% of sessions because our scaffold processes tool results in memory. TabPFN's meta-learned joint prior cannot opt out of a single feature the way XGBoost's axis-aligned splits can, and the out-of-distribution zero acts as a strong misdirected signal. XGBoost prunes the feature and the SLM detector reads a rendered text summary in which the same count is one of many fields rather than a joint-posterior dimension. The frozen Qwen judge also retains discriminative performance on these sessions. Overall, three of four detector-family representatives retain discriminative mean AUROC under the scaffold+LLM swap. Joint-prior tabular models are the class most sensitive to any single feature whose baseline value shifts across agents.

N COMPUTE COST

This appendix estimates the cost of reproducing the reported ACE capture, Qwen judge sweep, and SLM training runs. Additional cross-agent and exemplar-selection runs are accounted for separately below. Representation ablations and other exploratory runs are excluded. Session-capture Bedrock spend and SLM training GPU-hours are empirical, aggregated from per-session inference-usage records and per-adapter training telemetry respectively. Token counts for the three-model Qwen prompt comparison are aggregated from provider-reported usage for the returned responses. All dollar figures are approximate and use the recorded AWS Bedrock rates for each model and region, or August 2026 EC2 rates in `us-east-1`. Rates are point-in-time and subject to change.

N.1 SESSION CAPTURE

All 4,047 ACE sessions run one Claude Haiku 4.5 agent per fresh Docker container. Container-level session duration is empirical from per-session wall-clock records: median 8.8 s, mean 14.9 s, p90 22.4 s, aggregating to 16.7 container-hours of serialized wall-clock over all captured sessions. Capture was parallelized across concurrent containers, so total elapsed wall-clock is shorter than the serialized figure.

Agent-side Bedrock token usage is aggregated from the inference-usage records emitted by the CLI's inference backend on every session turn: 528,158 base input tokens, 3.10 M output tokens, 937.99 M cache-read input tokens, and 43.26 M 5-minute cache-write tokens. Cache-read dominates because the CLI's fixed tool-schema block is cached across turns within a session and re-read on every subsequent turn.

At published Bedrock rates for Claude Haiku 4.5 ($1.00/M base input, $5.00/M output, $0.10/M cache read, $1.25/M 5-minute cache write; `us-east-1`), total agent-side capture spend is approximately $164.

N.2 LLM-JUDGE SWEEP

The LLM-judge sweep reported in the paper (baseline and OWASP-grounded templates $\times$ shot counts $K \in \{0, 1\}$ $\times$ three views on ACE), including both fixed-reference and family-held-out evaluations, totals 211,386 returned judge responses across three Qwen models, including 211,175 valid scores. Per-model response counts and estimated costs are in Table 28. The accounting covers returned responses for the configurations in Tables 17 and 19; unsuccessful requests and retries are not separately costed.

Token usage. Provider-reported usage totals 1,596,215,947 input tokens and 12,428,934 output tokens across both prompt templates, both shot counts, and all three views.

Cost. At the recorded September 2026 Bedrock rates, the estimated LLM-judge sweep cost is approximately $288. Qwen3-235B uses `us-west-2` ($0.22/M input, $0.88/M output), Qwen3-80B uses `us-east-1` ($0.15/M input, $1.20/M output), and Qwen3-32B uses `us-east-2` ($0.15/M input, $0.60/M output). Dollar amounts are token-based estimates, not AWS invoices.

Table 28: **Estimated LLM-judge sweep cost by model.** Response and token counts are provider-reported; dollar figures use the September 2026 model and region rates in §N.2.

Model	Responses	Input (M)	Output (M)	Cost
Qwen3-235B-A22B-2507	70,542	536.8	4.02	$122
Qwen3-Next-80B-A3B	70,542	536.8	4.56	$86
Qwen3-32B	70,302	522.5	3.86	$81
Total	211,386	1,596.2	12.43	$288

N.3 SLM LoRA TRAINING

The 108 LoRA adapters for ACE's ID and OOD evaluations (three base models $\times$ three views $\times$ 12 folds) required 368.6 GPU-hours in aggregate, empirical from per-adapter epoch-timing telemetry summed across these adapters. Per-base means: Qwen2.5-3B 3.12 h/adapter (36 adapters, 112.3 h), Qwen2.5-7B 3.76 h/adapter (36 adapters, 135.4 h), Llama-3.1-8B 3.36 h/adapter (36 adapters, 121.0 h). Every adapter trains on a single 40 GB A100 at the sequence cap described in Appendix K.1.

We report cost against an AWS `p4d.24xlarge` instance (8×A100 40 GB, on-demand $21.958/instance-hour in `us-east-1`) as the reference hardware. Two cost accountings bracket the range:

- **Fully packed (8-in-parallel).** $368.6\,\text{GPU-h}/8\,\text{GPUs} = 46.1$ node-hours $\times$ $21.958 = approximately $1,012. This is the cost achievable when adapters run 8-in-parallel on one `p4d.24xlarge`, which reflects our operational configuration.
- **Fully serial (one active GPU per node-hour).** $368.6 \times \$21.958 =$ approximately $8,093. This is the upper bound when only one of the eight GPUs is active at a time.

The parallel figure reflects the operational cost realized in our runs. The serial figure is an upper bound.

N.4 CLASSICAL, DEEP-TABULAR, AND RULE-BASED DETECTORS

Aggregate CPU-time for AdaBoost, XGBoost, TabPFN, TabICL, and the ported Falco rule engine is under 5 CPU-hours across the full ID and OOD fold sweep, negligible on the same axis as the LLM and SLM figures.

N.5 AGGREGATE

The total estimated cost of ACE session capture and ID/OOD detector evaluation, including the known-family reference study, at the rates stated above is approximately:

- Session capture (Bedrock, Haiku 4.5; empirical): $164
- LLM-judge sweep (Bedrock, three Qwen models; metered tokens): ≈$288
- SLM LoRA training (`p4d.24xlarge` on-demand, 8-in-parallel): ≈$1,012
- Classical, deep-tabular, and rule-based: negligible

Estimated grand total: approximately $1,464 under 8-in-parallel SLM training accounting, rising to approximately $8,545 under fully-serial SLM accounting. All figures are approximate and are computed at the rates stated above. Rates are point-in-time and subject to change.

N.6 CROSS-AGENT AND EXEMPLAR-SELECTION STUDIES

The Qwen3-Next-80B-A3B cross-agent evaluation and exemplar-selection study add 21,488 scored responses: 1,409 ACE-XA session/fold scores, 4,027 zero-shot ACE scores, and 4,013 scores for each of the canonical and three random support sets. Across 21,588 targets, 96 are excluded by the local context-length check and four receive provider context-limit errors. There are 21,492 API attempts in total. The scanner comparison reuses cached scores and adds no inference cost.

Provider-reported usage totals 313,917,257 input tokens and 1,440,900 output tokens. At September 2026 `us-east-2` on-demand rates of $0.15/M input and $1.20/M output tokens, the estimated additional Bedrock cost is $48.82. This accounting uses metered token usage, though the dollar total remains an estimate rather than an AWS invoice.

The five same-fold Llama adapters use local V100 32 GB GPUs. Recorded training-plus-validation times are 1,290 s (A), 6,071 s (B), 712 s (D), 1,836 s (E), and 746 s (F), totaling 2.96 GPU-hours. This sum excludes model loading, tokenization, transfer inference, and pilots, and is not the elapsed duration of the parallel run. No cloud GPU rental charge was incurred. These additional runs are not included in the capture and ID/OOD evaluation costs above.