BEYOND THE DIFF: COST-AWARE AGENTIC TRAVERSAL FOR INDUSTRIAL GRADE SECURITY REVIEW

Anonymous authors
Paper under double-blind review

ABSTRACT

When state-of-the-art LLMs review code for security vulnerabilities, the true cost bottleneck is not finding bugs: it is adjudicating them to ensure minimal false positives. In production scanning, candidate findings from both probabilistic and deterministic scanners trigger multi-stage deliberation cascades (cross-vendor debate, rebuttal, and arbitration) where adjudicators read exhaustive repository context to prevent developer fatigue from false alarms. Strikingly, telemetry reveals that judges read 3.9× more changed files than candidate findings cite, burning over two-thirds of their token budget on unreferenced code. Grounded in reinforcement learning and sequential decision processes, we formulate cost-aware agentic scanning across two graphs: casting context retrieval as information acquisition over an *exploration graph*, and cascade termination as an optimal stopping policy over an *adjudication graph*. Along the first, anchoring judges strictly to candidate-cited evidence slashes judge inference costs by 44% to 58% across repository-disjoint replications. Along the second, sequential cost-aware stopping halts adjudication on a quarter of production runs, cutting their cost by 25.9% (15.4% across all traffic), while preserving 99.86% of final decisions and 99.2% of verified teacher keeps. Crucially, our evaluations reveal that cost optimization is not a free lunch: cited-file anchoring shifts verdicts beyond the judge's own repeat variability, while agentic navigators fail to outperform character-capped random retrieval. This study delivers the first end-to-end empirical decomposition of cost-fidelity dynamics in deployed agentic review, proving that context pruning, policy-driven stopping, and security correctness operate as distinct, non-interchangeable dimensions of production scanner design.

1 INTRODUCTION

As foundation models are mobilized to secure the global software supply chain, automated vulnerability scanning confronts a stark operational reality: finding potential bugs is no longer the bottleneck: adjudicating them without bankrupting the review pipeline is. Classical scanners traverse code property graphs with static taint queries (Yamaguchi et al., 2014); learned detectors supervise vulnerability classifiers over syntax and control flows (Zhou et al., 2019); and recent agentic architectures let language models explore repositories on demand (Li et al., 2025; Wang et al., 2024; Guo et al., 2025; Charoenwet et al., 2026). All of them produce an abundance of candidate findings. Yet in production, raw detection is plagued by false alarms. If an automated scanner inundates developers with spurious alerts, developers simply stop reading them (Xiong & Zhang, 2026; Du et al., 2026). Enterprise scanners must therefore make a critical second decision: *which proposed findings are genuine and severe enough to keep?* On runs that trigger this adjudication, verifying candidates accounts for the overwhelming majority of the deployment's model inference spend.

To filter candidate findings, deployed agentic scanners orchestrate sophisticated multi-stage deliberation cascades. In the production system we analyze, specialized *lenses* propose candidate *findings* with cited code locations. An *adjudication cascade* then puts each candidate on trial: two adversarial judges from different foundation model families argue for and against exploitability, a reconciler resolves their dispute, and subsequent challenge, rebuttal, and arbitration stages progressively refine

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.

the verdict. Every adjudicator reads broad repository context, and the organization pays for every token across every deliberative round.

Adjudicators receive broad context because true exploitability rarely lives in isolation: a missing authorization check matters only if an exposed caller reaches it, while a sanitization helper in another file can instantly refute a code injection claim. Consequently, the scanner renders tiered whole-PR contexts for every adjudicator. But this introduces a fundamental economic challenge: *how much context must an adjudicator read, and how many adjudication rounds must run, when every observation carries a token price?* Because an adjudicator cannot know whether a remote file matters without paying to ingest it, this is a sequential decision under partial information.

We organize the scanner's operational choices across two complementary graphs (Figure 1): the *exploration graph* over changed files linked by symbols and modules, and the *adjudication graph* over sequential deliberation stages. On each graph, we pursue the *golden path*: the cheapest possible execution trajectory that leaves the production scanner's ultimate decisions unchanged.

Using comprehensive telemetry, logged production verdicts, and prompt reconstructions from pipeline source, we study both interventions in an active enterprise deployment. Adjudicators read 3.9× more changed files than candidate findings cite, with cited evidence accounting for barely a third of the prompt. On the exploration graph, anchoring judges strictly to candidate evidence slashes inference cost by 44% to 58% across repository-disjoint replications; yet it changes 3 to 4 percentage points more verdicts than temperature-zero repeat variability, while agentic navigators fail to beat character-capped random retrieval. On the adjudication graph, a sequential cost-aware stopping rule halts deliberation early on a quarter of test runs, cutting their cost by 25.9% (15.37% overall) while matching 99.86% of candidate decisions and 99.2% of teacher keeps; in contrast, an untrained confidence threshold saves 19.26% but recklessly drops four teacher keeps across three repositories.

Our fidelity benchmark is the deployed adjudicator itself. Our goal is to measure behavioral preservation: whether pruning context or truncating deliberation alters the decisions the organization relies upon. We show that while massive cost reductions are achievable, cost optimization in agentic review is not a free lunch: context, stopping, and security correctness represent distinct dimensions that require independent empirical validation.

Contributions.

1. **Production Measurement.** We present the first end-to-end cost decomposition of a deployed multi-model agentic scanner by execution path, revealing that unreferenced code accounts for over two-thirds of adjudicator reading (Sections 3 to 4).
2. **Context Pruning & Navigator Diagnostics.** Replaying production judges across repository-disjoint cohorts, we establish the 44% to 58% cost reduction of cited-file anchoring against same-prompt variability, and provide the first empirical evidence that LLM graph navigators struggle to outperform simple random context baselines (Section 6).
3. **Cost-Aware Early Exits.** We demonstrate that sequential logistic stopping prunes a quarter of deliberation runs and saves 25.9% of their cost with near-perfect keep retention (99.2%), while exposing the severe vulnerability-drop risks of naive confidence thresholds (Sections 7 to 8).

2 Related Work

Routing and cascades. Learning to defer dates to rejection learning with an external decision-maker (Madras et al., 2018). Its LLM incarnations route or escalate among models (Chen et al., 2023; Ong et al., 2024; Feng et al., 2025; Jitkrittum et al., 2026); Dekoninck et al. (2025) unify routing and cascading, and Moslem & Kelleher (2026) survey the space. Confidence-based deferral has known failure modes (Jitkrittum et al., 2023), and common baselines matter (Li et al., 2026); we therefore compare our early exits with untrained confidence and unanimity gates on a deployed adjudication cascade.

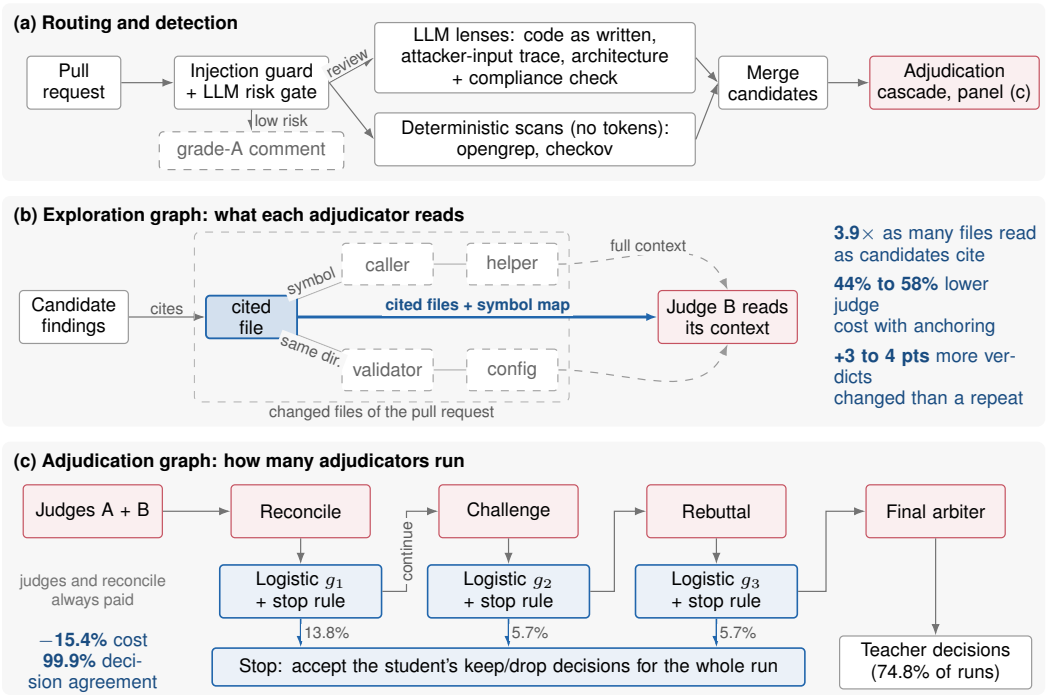

Figure 1: Overview. **(a)** An injection guard and a two-model risk gate send low-risk pull requests to a grade-A comment. The others go to three LLM lenses and a compliance check, with token-free opengrep and checkov scans in parallel; every candidate then enters adjudication. **(b)** The exploration graph links a pull request's changed files by symbols and directories. The production context gives the judge every changed file behind a symbol map; cited-file context (blue) keeps the symbol map and only the files the candidates cite. **(c)** The adjudication cascade with logistic early-exit classifiers. Their score margins and estimated remaining cost determine whether to stop or continue; percentages are shares of held-out test runs. Red marks costly adjudication calls, blue our policies. Statistics are from Sections 4 to 7.

Judging and selective evaluation. LLM judges (Zheng et al., 2023) can be distilled into smaller models (Kim et al., 2024; Zhu et al., 2025) or combined with selective escalation. Jung et al. (2025) give human-agreement guarantees under their framework; our replays measure agreement with a production adjudicator instead. Programmatic judges, in-context teaching and human escalation offer further economies (Huang et al., 2026; Wu et al., 2026; Fanconi & van der Schaar, 2025).

Adaptive retrieval. IRCoT interleaves retrieval and reasoning (Trivedi et al., 2023); FLARE and Self-RAG decide when to retrieve (Jiang et al., 2023; Asai et al., 2024); Adaptive-RAG sets retrieval depth by query complexity (Jeong et al., 2024). These methods retrieve for the model's own generation. Our exploration policies decide what a separate, more expensive adjudicator must read, price every token it reads, and are judged by whether the adjudicator's decision changes.

Program analysis and learned detection. Classical scanners encode vulnerability patterns as queries over a program representation. Code property graphs join syntax, control flow and data dependence into one structure that taint-style templates traverse (Yamaguchi et al., 2014), the model behind production query engines such as CodeQL and Semgrep. These analyses are precise where a specification exists but need hand-written sinks, sanitizers and propagators, and they need code that parses or compiles. Learned detectors replace the specification with supervision: Devign classifies functions from a composite graph (Zhou et al., 2019). Both families decide *whether* a location is vulnerable on a fixed input. Our question is orthogonal and comes after detection: given candidate findings, how much surrounding code must an adjudicator read, and how many adjudicators must run, before the keep/drop decision stops changing.

Hybrid and agentic scanning. Recent systems pair a model with an analysis engine. IRIS has an LLM infer taint specifications and then runs CodeQL, reporting more of its benchmark's vulnerabilities than CodeQL alone (Li et al., 2025). LLMDFA outsources parsing and path-feasibility checks to a parser and a theorem prover so that data-flow analysis needs no compilation (Wang et al., 2024). Agents instead explore on demand: RepoAudit walks data-flow facts along feasible paths with agent memory and a validator, and reports its cost per project (Guo et al., 2025); AgenticSCR adds security-focused semantic memory for pre-merge review and reports a shadow deployment (Charoenwet et al., 2026). These agents choose what to read to *produce* findings, and are scored on detection. Our exploration policies choose what a separate, more expensive adjudicator reads to *judge* findings already proposed, and are scored by whether the adjudicator's verdict changes. Sequential and multi-agent detection with confidence-based stopping (Tsai et al., 2025) is the closest analogue to our adjudication-graph policy, but it selects among detection strategies rather than truncating a deployed adjudication cascade.

Triage, evaluation and deployed practice. Because scanners over-report, a second line of work filters their output: LLM triage of static-analysis alerts measures false-positive reduction against cost (Xiong & Zhang, 2026; Du et al., 2026), and diagnostic evaluation separates a correct verdict from a correct explanation (Chen et al., 2026). CyberGym scores agents on proof-of-concept generation (Wang et al., 2025), a stronger bar than any agreement metric, including ours. Pan et al. (2026) study deployed agents through practitioner reports; we contribute telemetry and replays from one deployment. Robustness of cooperative multi-agent policies under induced faults is studied separately by Mguni et al. (2026), whose switcher decides when to perturb an agent under a budget; our stopping rule decides when to stop paying for one, and neither transfers to the other.

3 Study Setting and Data

System. The deployment scans pull requests (Figure 1a). An injection guard and a two-model risk gate run first; pull requests the gate rates low-risk receive a grade-A comment and no further review. The rest go to a fixed lens set: three analytic lenses (code as written, attacker-input tracing, and architectural dependencies across trust boundaries) and a compliance check, each reading the whole diff, with large pull requests split into chunks that every lens reads. Deterministic opengrep (OWASP Top-10) and checkov (infrastructure-as-code) scans run in parallel on the changed code and cost no model tokens. All candidates are merged and enter the adjudication cascade of Figure 1c. Both judges and most later adjudication stages receive the same code block: a tiered rendering of each changed file, keeping changed hunks and their enclosing functions, eliding other bodies to signatures, and prefixing a symbol map. This tiered context was enabled for 99.9% of judge calls in our window. An opt-in agentic pass that lets a model fetch extra files exists but was disabled in every one of them. Claude-based decision roles, including Judge B, run at temperature zero.

Why judges and adjudicators. Detection lenses seek candidate defects; adjudication is intended to reduce unsupported reports. Judge A argues for candidates and Judge B, from a different vendor, against them, providing multi-vendor diversity across prompt evaluation. Reconciliation and confidence-based routing separate fast-tracked verdicts from challenged candidates; rebuttal and final arbitration refine the result. Section 7 measures when the later stages can be skipped without changing final decisions.

Why this system is the reference. We measure fidelity to the production cascade because it is the system the organization relies on: a cost intervention that silently changes its decisions is an unreviewed policy change. The cascade is maintained by human-feedback-driven prompt and policy maintenance; no model weights are trained. A scheduled pipeline collects organic developer replies to posted findings daily, and a weekly uniform sample of posted critical-, high- and medium-severity findings that security experts mark as true or false positives with a stated reason. Security engineers turn these judgments into reviewed changes to prompts, global invariants and per-repository security context. Fidelity is therefore measured against the adjudicator's own decisions, not against independent vulnerability labels.

Feedback and drift. The same loop makes the reference an evolving target. Each accepted change updates what the adjudicator reads and how it decides, so the prompts and context behind a logged

Table 1: Populations, all within one four-day window. Sizes are given as study design (replayed runs) or as a share of the population they are drawn from; absolute deployment volumes are withheld.

Population	Window	Question
Scheduled runs	Workload day	Workload and model tiers
Cost-bearing node events	Two days	Stage decomposition
Judge-B calls, one per run	Later two days	What adjudicators read
Exploration-graph replay and repeat, 2 × 120 runs	Later two days	Reading the cited evidence
Complete cascade paths	All four days	Adjudication-graph stopping

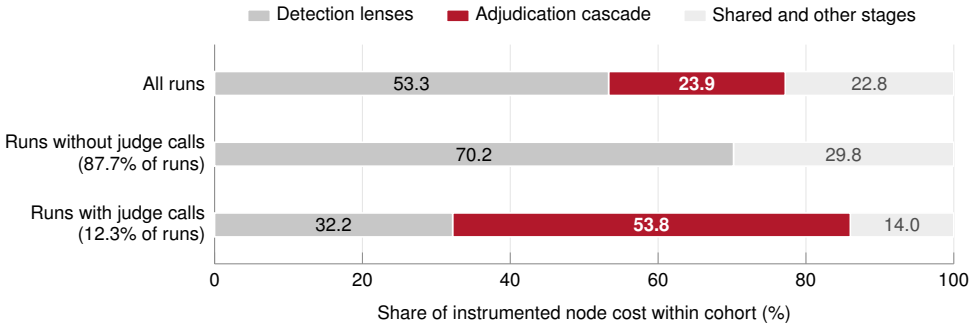

Figure 2: Where model cost goes, by execution path, as percentages of instrumented node cost within each cohort. In runs that invoke judges, adjudication (red) is the largest share; across all traffic, detection is. Runs without judge calls include completed negative scans, early exits and incomplete outputs. Absolute volumes and dollars are withheld.

call can *drift* from those in the current pipeline source. We therefore freeze one cohort, identify the observed settings by a configuration fingerprint (Appendix E), and separate two comparisons in Section 6: an independent repeat of the full-context replay, which isolates the judge's own run-to-run variability, and the full-context replay against the logged production call, which additionally captures operational drift.

Populations and disclosure. Each question uses its own population (Table 1), drawn from one four-day window; shares are not interchangeable across rows. Under the owning organization's disclosure policy we report *relative* quantities only: no absolute run, call or finding volumes, no dollar totals and no calendar dates. Costs therefore appear as percentages of a stated baseline, derived from the deployment's own list-price usage records rather than invoices. We give replay sample sizes and repository-bootstrap intervals where available; post-hoc subgroup point estimates are labeled separately. Under enterprise disclosure guidelines, we report relative performance metrics rather than absolute commercial volumes. Of the scheduled assignments on the workload day, 46.5% performed a full review, 42.3% were skipped by the risk gate and 10.5% short-circuited as incremental no-ops; 10.6% carry no cost record (Appendix D). Appendix E documents the observed settings by verified configuration fingerprint.

4 Where Adjudication Cost Goes

Stages. Detection accounts for 53.3% of instrumented node cost across the two-day cohort (Figure 2). Judges are invoked on 12.3% of those runs, and within that subset adjudication instead accounts for 53.8% of cost. Runs without judge calls are completed scans that left no candidate, early exits and incomplete runs; their detection spend, 38.9% of the cohort total, locates cost on an execution path and is not evidence of avoidable work. Appendix C gives per-node shares.

What adjudicators read. Across the Judge-B calls of the later two days, one per run, the code block holds 3.9 times as many changed files as the candidates cite, on average. In 82% of calls at least one file in the block is cited by no candidate, and the cited files hold 32.6% of its file text. In

the two replay samples of Section 6 the block is 82% and 72% of judge prompt characters; both judges and the main cascade stages receive it. Most of what the judges read is code that no finding points to. Whether that code changes their decisions is an empirical question, which Section 6 tests by replaying one judge.

5 Cost and Fidelity Objectives

We evaluate policies that select context or stop an adjudication cascade. Let c_t be the recorded model charge for a paid call and Y^R the reference adjudicator's decision. A reference-relative design objective is

$$\min_{\pi} \mathbb{E}_{\pi}\left[\sum_{t} c_t\right] \quad \text{subject to} \quad \Pr_{\pi}(\widehat{Y} \neq Y^R) \leq \epsilon, \qquad \Pr_{\pi}(\widehat{Y} = 0 \mid Y^R = 1) \leq \delta. \tag{1}$$

Because security errors are asymmetric (missing a vulnerability risks operational compromise while an extra alert merely costs reviewer time), the second constraint isolates keep-retention from general agreement. We report both at measured operating points for cited-file selection and early stopping. Only the stopping rule chooses its operating point, against development-set targets.

The *exploration graph* (Figure 1b) has one node per changed file in the logged context, linked by shared symbols, module stems and directories (Appendix A). The *anchors* are the files cited by candidate evidence locations; the Anchor policy keeps exactly these, so it follows citations rather than edges. The *adjudication graph* is the ordered cascade of Figure 1c, and an early exit truncates its remaining suffix. Every file read and every stage run is paid at its recorded token charge.

6 Reading Less: Cited-File Context

In production, adjudicators receive tiered renderings of every changed file under the hypothesis that surrounding code reveals compensating controls or remote sinks. But does this broad context justify its token cost? We test whether an adjudicator can maintain decision fidelity when anchored strictly to the candidate findings' cited evidence.

Policies. Each policy selects file blocks from the production tiered context, preserving exact text, production order and the symbol map without reading unchanged code: **Full** uses every logged file (production context), while **Anchor** keeps only the candidate-cited files (retaining full context if none is cited, which occurred in one run). The exploration graph is a descriptive organization of these files, not an identified optimal traversal.

Replay design. We replay the cascade's Judge B (`claude-sonnet-4-6`) through the deployment's own endpoint, reconstructing its prompt from the available pipeline source. On three reconstruction checks, replayed prompt lengths match the logged calls to within 0.082% of tokens. The first sample holds the first 120 eligible runs of the later two days in hash order: the first run of each pull request, at most three per repository, and 1.4 runs per repository on average. The replication sample repeats the procedure on 120 runs from repositories disjoint from the first. Protocol, code and data hashes were frozen before each replay, under a spend cap. A second, separately frozen protocol then asked the judge again on the identical prompts: one more full-context call per run, and a fresh anchored call even where the anchored context equals the full one (22% of runs), whose verdicts the first replay had reused. Every repeated prompt matched its original call's input tokens exactly. All new inference stayed under four fifths of its caps.

Candidate-label coverage. Within the eligibility filters, neither sample is chosen by vulnerability type. Candidates carry 87 distinct specific CWE labels (53 and 74 per sample, 40 shared), spanning injection, path traversal, authorization, cryptography, deserialization, SSRF and resource exhaustion; 89% and 95% of runs name at least one specific CWE. These are proposed, not verified, vulnerability classes (Appendix A).

Fidelity reference and judge variability. Fidelity is agreement with the original full-context replay, and a verdict agrees only if both are equal and present. Asking again on the identical prompt

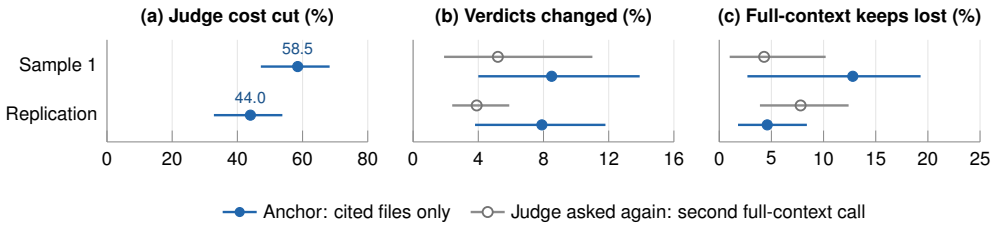

Figure 3: Anchoring the production judge on two repository-disjoint samples of 120 pull requests each. Points are estimates and bars 95% repository-cluster bootstrap intervals. **(a)** Reading only the cited files cuts the judge's cost by 44% to 58%. **(b, c)** Both markers are fresh, independent calls scored against the same full-context replay, so the open circle is the judge's own variability. Anchored verdicts change more often in both samples; kept findings are lost more often in the first sample, where over half of the losses fall in one run, and less often in the replication, where drops turn into keeps instead. Appendix Table 2 lists every rate, including the comparison with the logged production call.

measures the judge's own same-prompt repeat variability: the second full-context call changes 5.2% of first-sample verdicts and 3.9% in the replication, and loses 4.3% and 7.8% of full-context keeps. Even where anchored and full contexts are byte-identical, two fresh calls disagree on about 4% of candidates. The full replay's disagreement with the logged production call, 6.2% and 4.0%, also includes drift through the feedback loop of Section 3 and reconstruction error. Intervals use 2,000 repository-cluster bootstrap draws that resample paired comparisons together.

Anchoring cuts the judge's cost by 44% to 58%. Restricting Judge B strictly to cited evidence delivers dramatic savings (Figure 3a): recorded inference cost drops by 58.5% in the first sample (interval 47.2% to 68.3%) and by 44.0% in the replication (32.8% to 53.8%), both comfortably clearing pre-registered 30% bounds. Fresh calls reproduce both estimates to within 0.1 points. Yet this efficiency is not lossless: fresh anchored verdicts diverge from full-context replay on 8.5% and 7.9% of candidates, which is 3.3 and 3.9 percentage points higher than the judge's own repeat variability (paired intervals -3.1 to $+9.3$ and $+0.1$ to $+7.7$). Anchoring moves decisions beyond temperature-zero noise; in the replication, the excess interval strictly excludes zero.

Where the verdicts move. Critically, this decision drift operates in opposing directions across cohorts. In the first sample, anchoring loses 12.8% of full-context keeps (vs. 4.3% for repeat), with over half concentrated in one run. In the replication, anchoring loses fewer keeps than repeat (4.6% vs. 7.8%), but converts 8.4% of full-context drops into keeps (vs. 2.6% for repeat; paired excess 5.8 points, 1.4 to 10.3). The judge's instructions explain this duality: it drops candidates mitigated by compensating controls and validates candidates by tracing attacker data to sinks. Pruning uncited files can thus sever a true sink (losing a legitimate keep) or hide a compensating sanitizer (spawning an unsupported keep). The shift is structurally bidirectional, demonstrating why simple heuristic corrections fail.

Navigation and escalation controls. Do autonomous agentic navigators resolve this dilemma? Our pre-specified controls reveal a clear negative: neither cost-aware nor cost-free LLM navigators consistently outperform a simple pseudo-random file baseline clamped to the same character cap. Furthermore, relying on model self-escalation failed its pre-registered target (H2: 5.9% cost-cut lower bound vs. 10% threshold), losing 14 keeps while saving little. These findings provide a vital reality check: in industrial review, autonomous LLM traversal over repository graphs can easily inflate token overhead without outsmarting static citation anchoring (Appendix Table 3).

7 Stopping Sooner: Logistic Early Exits

If cited-file context prunes spatial redundancy across code files, early stopping prunes temporal redundancy across review stages. A deployed adjudication cascade operates like an adversarial court: two LLM judges debate candidate findings, a reconciler negotiates consensus, and specialized chal-

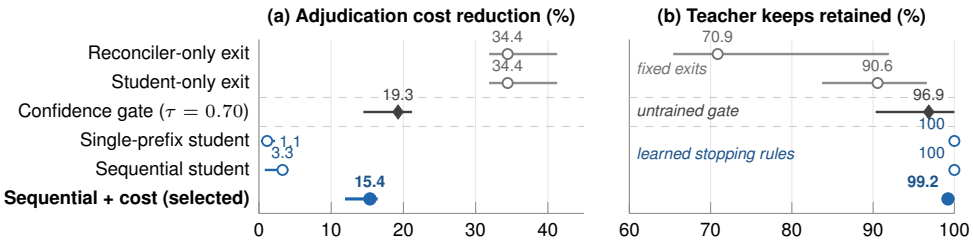

Figure 4: Cascade stopping on **all 210 held-out test runs**, relative to the full teacher cascade (0% reduction, 100% retention). Points are frozen policies; bars are 95% repository-bootstrap intervals where available (zero-error retention intervals are unavailable, not zero-width). Exiting after reconciliation saves a third of adjudication cost but drops many teacher keeps. The untrained confidence gate (diamond) saves more (19.3%) than learned stopping rules but misses 4 of 127 teacher keeps; learned rules retain almost every keep while saving less. The selected policy (filled) saves 15.4% and retains 99.2% of keeps. Retention concerns teacher keeps, not independent security quality. Table 4 lists all rates; Table 5 details early-exit subgroups.

lenge and arbitration stages deliver binding decisions. Yet in production, many findings settle early. Forcing every pull request through the complete five-stage cascade squanders expensive tokens on foregone conclusions.

Teacher and students. The reference is the full cascade of Figure 1c, and we take its final keep/drop decisions as our supervision target. Three lightweight logistic regression classifiers—termed *students* because they fit the teacher's verdicts—predict the final decision after reconciliation, challenge, and rebuttal. Candidate severity, a public-entry indicator, judge vote and confidence summaries, current verdict and confidence, unanimity, stage index, and candidate count provide 12 numerical features. Final labels and future costs are strictly excluded from inference. We use training-only normalization and class weighting, 1,200 gradient steps, learning rate 0.1, and L2 penalty 0.01. Crucially, no foundation model weights are fine-tuned.

Why a simple classifier. Instead of deploying another heavy language model to supervise the cascade, we test whether signals already emitted during deliberation can reliably indicate when debate is settled. Logistic regression requires zero additional model calls, evaluates in 28 microseconds per run, and provides transparent, auditable coefficients (Madras et al., 2018; Dekoninck et al., 2025). Its natural competitors are untrained heuristics reading the same signals: stage confidence thresholds and unanimity flags currently used in routing.

Stopping with a cost signal. At prefix t, let p_{tj} be the student's score for candidate j and $m_t = \min_j |p_{tj} - 0.5|$ the run's smallest margin. The controller stops if

$$m_t + w \log(1 + \widehat{C}_{\text{remaining},t}/C_0) \geq \eta, \tag{2}$$

and otherwise runs the next stage (C_0 is one dollar). Remaining cost $\widehat{C}_{\text{remaining},t}$ scales mean training stage cost per candidate by current candidate count. Stopping accepts student verdicts for the run, preserving batching. We evaluate $w \in \{0, 0.05, 0.10\}$ and $\eta \in \{0.20, 0.30, 0.40, 0.45, 0.49, 1.0\}$, selecting $(\eta, w) = (0.45, 0.10)$ on development data to maximize savings under 99% agreement and keep retention; cost-unaware control uses $(0.45, 0)$.

Untrained gates. Three untrained baselines stop at the first stage where, across all candidates, stage confidence reaches τ (*confidence*), judges are flagged unanimous (*unanimity*), or both, accepting that stage's verdicts. We select $\tau \in \{0.50, \ldots, 0.99\}$ on development data under the same 99% constraints ($\tau = 0.70$ chosen) and evaluate once on test under a separately frozen protocol.

Data and accounting. Of exported four-day paths, 97% are complete and unambiguous. Hashing by repository and day assigns 63% of runs to training (day 1), 9% to development (day 2), and 28% (210 runs, 18% teacher keep rate) to test (days 3 and 4, disjoint repositories). Evaluated costs use recorded node charges across adjudication stages only, excluding detection and local compute.

Result. The cost-aware policy halts after reconciliation, challenge, or rebuttal on 13.8%, 5.7%, and 5.7% of test runs, executing the full cascade on 74.8%. It matches 99.86% of candidate decisions, retains 99.2% of teacher keeps (one miss), and adds zero dropped candidates. Recorded adjudication cost drops by 15.37% (bootstrap interval 11.92% to 16.44%). Zero-cost stopping saves 3.27% with zero disagreements. The standalone student agrees less than the reconciler (83.86% vs. 90.63%), making escalation essential. One dominant repository contributes 65% of runs and the missed keep; outside it, the policy saves 3.15% with perfect agreement.

Where the saving comes from. The policy's early exits comprise 25.2% of test runs and 59.34% of baseline adjudication cost. Within them, cost falls by 25.9% with 99.6% agreement and 95.8% keep retention (the sole miss); full-cascade runs preserve teacher decisions, yielding $0.5934 \times 25.9\% \approx 15.37\%$ overall savings (Appendix Table 5).

Untrained gates save more and lose more keeps. This comparison highlights the hazards of uncalibrated cost cutting. An untrained confidence gate ($\tau = 0.70$) exits after reconciliation on 47.6% of runs, cutting cost by 19.26% (14.46% to 21.2%). Yet it drops 4 of 127 teacher keeps across three repositories (96.9% retention), failing the 99% development target; within early exits, retention collapses to 81.0%. Conversely, our cost-aware sequential student acts with surgical discipline: halting 25.2% of runs, saving 25.9% within them (15.37% overall), and missing just a single keep (95.8% subgroup, 99.2% overall retention). The unanimity gate misses development targets (95.7%) and defaults to full cascade. In security scanning, where unflagged exploits risk enterprise breach, learned sequential stopping protects the recall that heuristic gates discard.

8 Discussion and Limitations

Two savings with different denominators. Anchoring removes 44% to 58% of replayed Judge-B cost; cascade stopping removes 15.4% of all adjudication cost. These percentages are not directly comparable: Judge B is one of multiple calls per run, and both were measured on separate samples. While judges and later stages share tiered code blocks, later stages use prompt caching, shrinking marginal gains. The 15.37% all-run saving must be weighed against engineering overhead; the conditional 25.9% early-exit rate cannot replace all-traffic accounting.

Fidelity is not a safety margin. Measured variability does not subtract security risk. Asked again on identical prompts, the full-context judge loses keeps in 5% and 10% of runs; fresh anchored calls lose keeps in 6% and 7% of runs and turn drops into keeps in 11% and 13%. Without a reliable fallback predictor, teacher-relative changes in either direction require independent security validation.

Limitations. Our reference is the production adjudicator, not independent ground truth: a lost keep is disagreement with a judge that may itself err. No replayed findings had completed human verification. Every policy selects among changed files as rendered; none fetches uncommitted repository code. We replayed one judge of several; anchoring lowers Judge B's agreement with the arbiter from 87.7% to 84.4% on replication. Replays cover two days (240 runs across disjoint repositories), and cascade stopping was evaluated on complete paths where one repository supplies 65% of runs.

9 Conclusion

As agentic LLMs transition from benchmarks into software engineering, cost efficiency and security fidelity must be co-designed. This paper delivers the first production-scale study decomposing where inference spend goes during agentic code review. Restricting adjudicators strictly to candidate-cited evidence slashes judge inference costs by 44% to 58%, yet induces real decision drift beyond baseline model variability, while autonomous graph navigators fail to surpass naive random retrieval. Concurrently, lightweight sequential early exits safely eliminate 25.9% of deliberation expenditure on stopped runs (15.37% across all traffic) while preserving 99.86% of final decisions and 99.2% of teacher keeps, dramatically outperforming heuristic confidence gates in recall safety. Cost reduction in agentic workflows is neither free nor purely algorithmic: context pruning, stage stopping, and security correctness must each be rigorously audited before taking the golden path to production.

AI USE STATEMENT

We used generative AI tools for tasks with required disclosure: designing experiments, developing the cost-fidelity framing and break-even condition, preparing analysis datasets from logged telemetry, and interpreting results. We did not use them to generate synthetic datasets, prove mathematical claims, translate, or support qualitative analysis. We also used them to identify and summarize related work, format references, suggest the title and structure, execute the read-only queries, replays and analyses reported here. Replays sent reconstructed production prompts only to the organization's own model endpoint, under hard spend caps. Code is checked by unit tests, protocols were hash-frozen before the evaluations they govern, and reported numbers were checked against frozen outputs. We have reviewed all AI-assisted work and take responsibility for the final content, including text, claims, figures, citations and numbers.

ETHICS STATEMENT

The subject system reviews authorized code within one organization. Measurements reported here are relative aggregates; no repository source, finding content, developer identity, comment text, absolute volume, dollar amount or calendar date is reproduced. Replay inputs were processed only on the authors' workstation and the organization's own model endpoint. Release of the underlying data is subject to the owning organization's approval, and anonymization alone does not grant that right. The application risks are asymmetric: a missed defect gives false assurance, and an unsupported report consumes reviewer attention. Neither the measurements nor the policies constitute a security guarantee.

REPRODUCIBILITY STATEMENT

Section 3 states the populations, cutoffs and eligibility filters; the read-only queries are preserved in the local research artifacts. Section 6 and Appendix A describe the replay: sample queries, prompt reconstruction check, frozen protocols for the replay and its repeat, code and data hashes, and the spend ledgers. Appendix B documents the cascade-stopping contract and the separately frozen protocol for the untrained gates. Absolute volumes, dollar amounts and dates are withheld from this paper under the owning organization's disclosure policy and retained with the private artifacts. The offline tooling is deterministic given its inputs, apart from model calls, and is covered by 199 tests. Production telemetry and replay outputs are confidential; external release requires authorization.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.11511.

Wachiraphan Charoenwet, Kla Tantithamthavorn, Patanamon Thongtanunam, Hong Yi Lin, Minwoo Jeong, and Ming Wu. AgenticSCR: An autonomous agentic secure code review for immature vulnerabilities detection. *arXiv preprint arXiv:2601.19138*, 2026. doi: 10.48550/arXiv.2601.19138. URL https://arxiv.org/abs/2601.19138.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023. doi: 10.48550/arXiv.2305.05176. URL https://arxiv.org/abs/2305.05176.

Zhuo Chen, Boyang Wang, Xiyue Zhang, Xiaoyun Xu, Ahmad-Reza Sadeghi, Stjepan Picek, and Lichao Wu. From verdict to diagnosis: Attributable security review of pull requests. *arXiv preprint arXiv:2608.25730*, 2026. doi: 10.48550/arXiv.2608.25730. URL https://arxiv.org/abs/2608.25730.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 12987–13010. PMLR, 2025. URL https://proceedings.mlr.press/v267/dekoninck25a.html.

Xueying Du, Jiayi Feng, Yi Zou, Wei Xu, Jie Ma, Wei Zhang, Sisi Liu, Xin Peng, and Yiling Lou. Reducing false positives in static bug detection with LLMs: An empirical study in industry. *arXiv preprint arXiv:2601.18844*, 2026. doi: 10.48550/arXiv.2601.18844. URL https://arxiv.org/abs/2601.18844.

Claudio Fanconi and Mihaela van der Schaar. Cascaded language models for cost-effective human-AI decision-making. *Advances in Neural Information Processing Systems*, 38, 2025. URL https://arxiv.org/abs/2506.11887.

Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A graph-based router for LLM selections. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.03834.

Jinyao Guo, Chengpeng Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. RepoAudit: An autonomous LLM-agent for repository-level code auditing. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 21083–21100. PMLR, 2025. URL https://mlanthology.org/icml/2025/guo2025icml-repoaudit/.

Tzu-Heng Huang, Shengqi Qiu, and Frederic Sala. Codifying the judge: Scalable evaluation via program distillation. *arXiv preprint arXiv:2607.22561*, 2026. doi: 10.48550/arXiv.2607.22561. URL https://arxiv.org/abs/2607.22561.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.naacl-long.389/.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7969–7992. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.495. URL https://aclanthology.org/2023.emnlp-main.495/.

Wittawat Jitkrittum, Neha Gupta, Aditya Krishna Menon, Harikrishna Narasimhan, Ankit Singh Rawat, and Sanjiv Kumar. When does confidence-based cascade deferral suffice? In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2307.02764.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient LLM inference. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2502.08773.

Jaehun Jung, Faeze Brahman, and Yejin Choi. Trust or escalate: LLM judges with provable guarantees for human agreement. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=UHPnqSTBPO.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.08491.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, Zhen Wang, and Shuyue Hu. LLMRouterBench: A massive benchmark and unified framework for LLM routing. *arXiv preprint arXiv:2601.07206*, 2026. doi: 10.48550/arXiv.2601.07206. URL https://arxiv.org/abs/2601.07206.

Ziyang Li, Saikat Dutta, and Mayur Naik. IRIS: LLM-assisted static analysis for detecting security vulnerabilities. *arXiv preprint arXiv:2405.17238*, 2025. doi: 10.48550/arXiv.2405.17238. URL https://arxiv.org/abs/2405.17238.

David Madras, Toniann Pitassi, and Richard Zemel. Predict responsibly: Improving fairness and accuracy by learning to defer. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://arxiv.org/abs/1711.06664.

David Mguni, Yaqi Sun, Haojun Chen, Wanrong Yang, Amir Darabi, Larry Olanrewaju Orimoloye, and Yaodong Yang. Fault tolerant multi-agent learning with adversarial budget constraints. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. doi: 10.48550/arXiv.2508.08800. URL https://arxiv.org/abs/2508.08800.

Yasmin Moslem and John D. Kelleher. Dynamic model routing and cascading for efficient LLM inference: A survey. *Transactions on Machine Learning Research*, 2026. URL https://arxiv.org/abs/2603.04445.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. *arXiv preprint arXiv:2406.18665*, 2024. doi: 10.48550/arXiv.2406.18665. URL https://arxiv.org/abs/2406.18665.

Melissa Z. Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, Shu Liu, Tianneng Shi, Xiaoyuan Liu, Jared Quincy Davis, Emmanuele Lacavalla, Alessandro Basile, Shuyi Yang, Paul Castro, Daniel Kang, Koushik Sen, Dawn Song, Joseph E. Gonzalez, Ion Stoica, Matei Zaharia, and Marquita Ellis. Measuring agents in production. *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2512.04123.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.acl-long.557/.

Chung-Nan Tsai, Xin Wang, Cheng-Hsiung Lee, and Ching-Sheng Lin. A sequential multi-stage approach for code vulnerability detection via confidence- and collaboration-based decision making. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21151–21157. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1071. URL https://aclanthology.org/2025.emnlp-main.1071/.

Chengpeng Wang, Wuqi Zhang, Zian Su, Xiangzhe Xu, Xiaoheng Xie, and Xiangyu Zhang. LLMDFA: Analyzing dataflow in code with large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4181. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ed9dcde1eb9c597f68c1d375bbecf3fc-Abstract-Conference.html.

Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. CyberGym: Evaluating AI agents' real-world cybersecurity capabilities at scale. *arXiv preprint arXiv:2506.02548*, 2025. doi: 10.48550/arXiv.2506.02548. URL https://arxiv.org/abs/2506.02548.

Yu Wu, Shuo Wu, Ye Tao, Yansong Li, and Anand D. Sarwate. From deferral to learning: Online in-context knowledge distillation for LLM cascades. *arXiv preprint arXiv:2509.22984*, 2026. doi: 10.48550/arXiv.2509.22984. URL https://arxiv.org/abs/2509.22984.

Yunpeng Xiong and Ting Zhang. Sifting the noise: A comparative study of LLM agents in vulnerability false positive filtering. *Proceedings of the ACM on Software Engineering*, 3(ISSTA), 2026. doi: 10.1145/3832100. URL https://arxiv.org/abs/2601.22952.

Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy*, pp. 590–604. IEEE, 2014. doi: 10.1109/SP.2014.44. URL https://ieeexplore.ieee.org/document/6956589/.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 36, 2023. URL https://arxiv.org/abs/2306.05685.

Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in Neural Information Processing Systems*, volume 32, pp. 10197–10207, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2310.17631.

A Exploration-Graph Replay Detail

Table 2: Exact rates behind Figure 3; all entries are percentages or percentage points. Unless marked, the reference is the original full-context replay. *Judge asked again*: a second full-context call. *Excess*: fresh anchor minus judge asked again, with paired repository-bootstrap intervals. *Changed*: verdicts that differ from the reference or are missing. *Lost*: reference keeps not kept. *Drops kept*: reference drops turned into keeps. The last row uses the logged production call as its reference. Cost cut is recorded Judge-B cost relative to full context. Each sample replays 120 pull requests from disjoint repositories.

	Sample 1				Replication			
Comparison	Cost cut	Changed	Lost	Drops kept	Cost cut	Changed	Lost	Drops kept
Judge asked again	-	5.2	4.3	2.4	-	3.9	7.8	2.6
Anchor, fresh call	58.5	8.5	12.8	4.9	44.1	7.9	4.6	8.4
Excess (points)		+3.3	+8.6	+2.4		+3.9	−3.2	+5.8
95% interval		−3.1, 9.3	−4.4, 16.3	−0.2, 6.8		0.1, 7.7	−7.5, 0.5	1.4, 10.3
Anchor, original	58.5	8.7	14.4	4.3	44.0	7.2	8.7	6.5
Full vs. logged call	-	6.2	7.5	4.2	-	4.0	7.7	3.0

Table 3: Additional navigation and escalation controls from the frozen five-arm replay. Cost cut includes navigator calls; the random arm is matched to the navigator's added character cap, not to dollars or tokens. "Changed" is disagreement with the full replay; "lost" is the percentage of full-context keeps not retained. The first-sample escalation policies were selected after inspecting that sample and are exploratory; the replication excludes its repositories.

	Sample 1			Replication		
Policy	Cost cut	Changed	Lost	Cost cut	Changed	Lost
Anchor, cited files	58.5	7.9	14.4	44.0	7.2	8.7
Navigator, cost-aware	53.7	8.3	14.4	37.2	6.9	9.2
Navigator, no cost	52.1	8.3	15.5	37.0	7.1	8.7
Random, character cap	55.8	7.5	14.4	41.5	6.8	8.7
Escalate on request	27.3	2.9	4.3	14.2	5.6	6.4
Escalate, low anchor share	13.2	2.3	1.6	20.9	6.1	7.8

Navigation hypotheses. The replication met H1: the Anchor cost-cut lower bound exceeded 30%. It met H3 at the point-estimate threshold: escalation-on-request disagreement was 5.6%, no more than the full-replay baseline plus two points. It did not meet H2: escalation-on-request lost fewer keeps than Anchor (14 versus 19), but its cost-cut lower bound was 5.9%, below the pre-specified 10% threshold. The navigator's linked-file choices showed no consistent advantage over

the character-cap random control, showing that graph-based exploration adds token cost without outperforming random retrieval or static citation anchoring.

Eligibility and sampling. A run is eligible if it is the first run of its pull request in the window, has exactly one non-abstained Judge-B call with tiered context enabled, and has one final-arbiter decision set. Runs are ordered by a salted SHA-256 hash of their identifier and taken in order, at most three per repository, until 120 are selected. The replication applies the same query to repositories absent from the first sample. The reconstruction check uses three runs from an earlier day, outside both samples.

Prompt reconstruction. The judge rules, the Judge-B role prompt, the candidate serialization, the policy and untrusted-repository sections, and the JSON parser are loaded from the pipeline source available to us by syntax-tree evaluation, without executing the pipeline; replay code maps each parsed decision to `valid`, `invalid` or `missing`. The prompt has the production layout: role prompt and rules, the code block, policy sections, then the candidates. Replayed input tokens differ from the logged calls by at most 0.082% on the three checks (one matches exactly; the other two differ by under 0.1%), and verdicts agree on five of the six checked candidates, confirming high-fidelity prompt reconstruction.

Graph construction. The logged code block is split losslessly into its symbol map and per-file blocks; concatenating the parts restores the block byte for byte. Its nodes are the pull request's changed files as production rendered them: no policy can reach a file outside that block. A file links to another when it names a symbol defined there, or the other file's module stem; files in one directory are linked both ways. The anchored context keeps the symbol map and the cited files' blocks in production order.

Candidate-label coverage. CWE labels are those the detection stage attached to candidates, taken from the normalized candidate records the judge receives; they describe what was proposed, not verified vulnerability classes. Across both samples the candidates carry 87 distinct specific CWE identifiers (53 in the first sample, 74 in the replication, 40 in both). The most frequent specific classes are SQL injection (CWE-89), OS command injection (CWE-78) and cross-site scripting (CWE-79) in the first sample, and improper restriction of a communication channel to intended endpoints (CWE-923), SQL and OS command injection in the replication. About half of candidates carry only the generic research-concepts label CWE-1000 (46% and 57%), and 4% and 3% carry no label. At least one candidate names a specific CWE in 89% and 95% of runs.

Spend. An append-only ledger records the cost of every model call, and a budget reservation stops each replay before its cap. Rate-limit rejections are charged zero. All other failed attempts are charged at conservative estimates, about 5% of total replay spend: handshake failures in the first sample's reconstruction check, recorded before the runner gained its zero-charge rule for requests that cannot have reached the model; the in-flight calls of one replay interrupted by a network outage; and replication timeouts and one connection error. The two samples used similar budgets (52% and 48% of total replay spend), and total spend stayed under three quarters of the authorized cap. The repeat replay had its own hard caps and ledgers; all of its calls succeeded, none was charged at an estimate, and it used under four fifths of its caps.

Additional measures. Anchoring reduces judge input tokens by 60.6% and 47.2% in the two samples. The dollar reductions are smaller because output tokens, priced five times higher, do not shrink: they rise 1.1% and 0.3%. Agreement with the final arbiter falls from 86.0% to 83.8% in the first sample and from 87.7% to 84.4% in the replication.

Missing verdicts. Missing or unparseable verdicts are 0.8% of original first-sample verdicts and 0.06% in the replication, counting reused outputs, and 1.1% and 0.1% of repeated verdicts. The protocols count a missing verdict as a disagreement. The frozen evaluator of the original replay compared verdict categories, so two missing verdicts matched; it reported 7.9% first-sample anchor disagreement, which becomes the 8.7% of Table 2 under the protocol's convention (`missing_sensitivity.py`, same bootstrap). The replication and every keep-loss count are unaffected. Both frozen outputs are preserved.

Table 4: Operating points behind Figure 4, on the 210 held-out test runs. Cost is recorded adjudication-stage cost indexed to the full teacher (100). Misses are teacher keeps the policy drops; they are not independently verified vulnerabilities. All other entries are percentages.

Policy	Cost index	Reduction	Disagreements	Missed keeps
Full teacher	100.0	-	0.0	0.0
Reconciler only	65.6	34.43	9.4	29.1
Student only	65.6	34.43	16.1	9.4
Confidence gate, $\tau = 0.70$	90.7	19.26	0.58	3.1
Unanimity gate (infeasible; full cascade)	100.0	0.00	0.0	0.0
Unanimity + confidence, $\tau = 0.70$	90.8	9.25	0.58	3.1
Single-prefix student	98.9	1.13	0.0	0.0
Sequential, no cost	96.7	3.27	0.0	0.0
Sequential, cost-aware	94.6	15.37	0.14	0.8

Identical-context reuse and repeat calls. Anchor equals full context in 22% of runs per sample (10% and 7% of candidates). The original replay reused the full-context verdicts there, so those candidates agreed by construction, and each policy was charged the call it would need. The repeat replay (`run_repeat_replay.py`) calls both arms independently on every run after checking that the prompt contract, the records and every rebuilt context hash equal the originals. On the byte-identical runs, two fresh calls disagree on 3.8% and 4.0% of candidates. Fresh anchored verdicts differ from the original anchored verdicts on 3.7% and 3.5% of candidates. The repeat ran after the original replays against the same model name, so any serving-side change in between counts as variability here.

B Cascade-Stopping Experiment Detail

Untrained gates. `marta_baselines.py` implements the gates of Section 7 under a protocol frozen before any gate was evaluated on any split; it verifies the corpus hash and that every split equals the frozen split manifest. On the 67 development runs the confidence gate saves 25.1%, 13.2%, 6.8% and 4.4% at $\tau = 0.50$, 0.60, 0.70 and 0.80, with keep retention of 71.0%, 92.8%, 100% and 100%, so the selection is $\tau = 0.70$. The unanimity gate saves 8.6% on development data but retains 95.7% of keeps, below the target. On test, the confidence gate's exact-run agreement is 98.1%, it keeps none of the teacher's drops, and its four misses fall in four runs from three repositories. The dominant test repository's 136 runs save 10.6% with 97.6% keep retention; the other 74 runs save 6.9% with 95.3%.

Post-hoc decomposition by the frozen stop decision. Table 5 partitions all test runs by whether the existing cost-aware policy exits before the final teacher stage. Membership uses only the saved `stopped_after` field: every non-teacher exit is included, regardless of its realized savings or verdict correctness. No model, threshold or split is changed. Cost reduction is the ratio of summed saved cost to summed baseline cost within each group, not the mean of per-run percentages. Agreement is candidate-weighted and keep retention uses each group's own teacher keeps. Matching the teacher on the full-cascade group is by construction. These post-hoc point estimates have no new subgroup uncertainty interval or independent replication. The script `stopping_subgroups.py` reproduces the decomposition from frozen predictions and checks the cost-weighted identity.

Table 5: Frozen-policy decomposition; all entries are percentages. Run share and baseline cost share use all test runs as denominator. Savings and fidelity use the row's own population. The subgroup's 25.9% saving must not be combined with the all-run fidelity.

Population	Run share	Baseline cost share	Cost reduction	Decision agreement	Keep retention
All test runs	100.0	100.0	15.37	99.86	99.21
Early exits	25.2	20.7	25.9	99.6	95.8
Full cascade	74.8	79.3	0.0	100.0	100.0

Logged paths and exclusions. The export selects the first scheduled run per repository and pull request in the four-day window, before eligibility. No source code, finding titles, rationales or developer identities enter the student export, and candidate identities are hashed within each run. Of the exported run paths, 2.8% lack decision payloads and a further 0.2% lack a complete or unambiguous decision set or a positively priced final stage. The remaining 97% have two judges, one occurrence of each later stage, matching candidate sets, ordered timestamps and a completed terminal event. The repository and day intersection then assigns the paths it retains to the splits below.

Split	Window	Share of runs	Candidates per run	Teacher keep rate
Train	First day	63.2%	3.42	24.2%
Development	Second day	8.9%	5.10	20.2%
Test	Last two days	27.9%	3.30	18.3%

Teacher identity and reference. The recorded roles use OpenAI and Anthropic models across verified prompt configurations documented in Appendix E. The complete final candidate verdict serves as the reference for behavioral evaluation, measuring student fidelity to the deployed cascade.

Student fit and cost estimation. Each prefix student has 12 numerical features: candidate severity, public-entry indicator, judge valid-vote fraction, mean confidence, confidence missingness, current verdict, confidence and missingness, unanimity, log candidate count, prefix index, and judge disagreement. All preprocessing and model weights use training data only. Final teacher labels are training targets, never inference features. The remaining-cost estimate at a prefix is the sum of remaining recorded training cost divided by training candidate count, multiplied by current run candidate count. It is constant for a given prefix and candidate count, not an independently learned run-specific cost predictor or causal per-candidate price. At inference, the stopping rule uses this estimate in its action score. Separately, development selection of the weight and cutoff maximizes savings computed from logged development charges. Test evaluation also uses logged charges; no realized future test cost is available to the controller when choosing an action.

Cost-aware stopping policy formulation. Student weights are trained from teacher labels, while the margin-plus-log-cost rule balances prediction certainty with remaining cost expectations. Its weight and cutoff are tuned on development data over the grid in Section 7. Both no-cost and cost-aware variants satisfy development constraints; on test, the cost-aware rule delivers an additional 12.1 percentage points of cost reduction while conceding only a single candidate disagreement out of 694.

Call accounting and telemetry detail. Relative to the teacher's recorded calls, the single-prefix, no-cost sequential and cost-aware sequential policies retain 98.2%, 96.0% and 93.3%; reconciler-only and student-only exits retain 65.8%. Stage stop rates differ from call rates because some stages contain multiple calls and others are no-ops. Positive-call records require exact model-key price coverage and consistent totals. Most test rebuttal records lack usage and satisfy a no-survivor, unchanged-decision no-op contract. Many switch-stage records report zero calls; their decisions and confidence match the reconciler, but the tracking implementation can also emit zero usage after an invocation with absent response metadata. All figures represent recorded list-price replay accounting based on logged pipeline usage.

Controls, uncertainty, and timing. The student-only baseline misses 9.4% of teacher keeps and keeps 17.6% of teacher drops; its 83.86% agreement is lower than the reconciler-only baseline's 90.63%. The random controller draws stops using development-prefix rates and is evaluated with 20 fixed seeds, averaging 4.72% recorded savings, 97.26% agreement and 98.39% teacher-keep retention. It is not an equal-test-spend control. All quality and cost intervals resample repositories jointly. The cost-aware agreement interval is 99.73% to 100%, and its teacher-keep interval is 98.94% to 100%, confirming high retention across repository clusters. About 64.8% of test runs come from one repository, which contributes the sole missed keep and 78.7% of savings. Local student and controller evaluation takes about 28 microseconds per run on the test host, demonstrating negligible computational overhead.

Table 6: Cohort split behind Figure 2, the two-day instrumented cohort. Relative cost per run is indexed to the cohort mean (1.00). The cascade dominates the judge-invoking subset; detection dominates the aggregate. Both subsets are measured on the same cohort and sum to the total.

Cohort	Runs	Cost share	Cost/run	Detection	Adjudication
All instrumented runs	100%	100%	1.00	**53.3%**	23.9%
no judge call	87.7%	55.5%	0.63	70.2%	0.0%
judge call	12.3%	44.5%	3.61	32.2%	**53.8%**

Table 7: Share of instrumented node cost over all security-review runs in the three-day window, from per-node usage records. Several nodes run per candidate or per lens. The two-day cohort of Table 6 excludes the first day, on which a fourth detection arm was still live.

Node	Share
Detection lenses	52.20%
Cache priming	11.89%
Dual judges	10.15%
Coverage matrix	9.43%
Reconciliation	5.98%
Final arbiter	4.91%
False-positive challenge	3.71%
Change relevance	0.80%
Rebuttal	0.62%
Risk classifier	0.20%
Additional scan	0.10%
Summary	0.02%
Adjudication cascade (5 nodes)	25.36%

Provenance. The protocol precedes student fitting and evaluation. Corpus, query, protocol, training code, checkpoint, development-grid results, per-run traces and a separate recorded-call audit are retained. Checkpoint reload reproduces scores exactly. The experiment needed no new model calls and made no production changes. Raw artifacts stay private and require approval for external release.

C Per-Stage Cost Detail

For comparison, an earlier development study put the adjudication cascade at 52.9% of recorded cost and the detection lenses at 27.6%; its stage subtotal differed by 0.7% from its run-level total. Those shares are reproduced by the production judge-invoking cohort (53.8% and 32.2%) and are inverted by the workload as a whole, which is the distinction Section 4 draws.

D Cohort and Per-Tier Cost Detail

E Recovered Deployment Configuration

Telemetry records runtime parameters and prompt configurations governing scanner behavior. The scheduling event logs the configuration that determines execution, and that configuration is constant by measurement: apart from per-pull-request identity fields, the two experiment arms and one field that is null throughout, every configuration key holds exactly one value across every scheduled run in the cohort. No pipeline-behavior setting varies. A SHA-256 fingerprint over the sorted constant settings uniquely identifies the verified cohort configuration and is retained with the private artifacts.

The recovered role assignments show the cascade is multi-vendor: judge A `openai-gpt-5.2`, judge B `claude-sonnet-4-6`, final arbiter `claude-opus-4-8`, risk-gate classifier `openai-gpt-5-mini`, and coverage matrix and summary `claude-sonnet-4-6`. The risk gate and incremental review are enabled, and deterministic scoring runs in shadow mode.

Table 8: One day of scheduled production reviews, observed at a fixed cutoff, as shares of each tier's assigned reviews. Cost is recorded whole-pipeline model cost from the deployment's own instrumentation, not an invoice; relative mean cost is over costed runs only and indexed to the cheap tier (1.00).

	Cheap tier	Mid tier	Premium tier
Share of assigned reviews	20.7%	58.9%	20.4%
Reviews per distinct PR	2.20	2.12	2.20
Completed at cutoff	99.9%	99.5%	97.8%
Cancelled at cutoff	0.1%	0.5%	2.2%
Full reviews	48.1%	45.8%	46.8%
Risk-gate skips	39.9%	43.7%	40.7%
Incremental short circuits	11.9%	10.1%	10.3%
Runs missing cost	12.0%	10.1%	10.4%
Share of recorded cost	8.3%	48.3%	43.3%
Relative mean cost, costed runs	1.00	1.99	5.16

Table 9: Cost of the first eligible completed full review per PR, restricted to the repositories represented in all three arms, indexed to the cheap tier (1.00). Populations still differ in which PRs they contain, so these are descriptive conditional differences, not causal per-model prices.

Tier	Share of PRs	Relative mean	Relative median	Mean changed files
Cheap	21.8%	1.00	1.00	16.7
Mid	56.8%	2.12	1.52	14.1
Premium	21.4%	5.20	5.65	13.9

Two operational characteristics characterize this production cohort: a second experiment arm assigns its treatment to 99.8% of scheduled runs, and the tier allocation is naturally weighted (58.9% mid tier against 20.7% cheap and 20.4% premium on the workload day). We report these as observed empirical workload characteristics across tiers.

Table 10: Stage composition of a judge-invoking run by detection tier, three-day window. Relative costs are indexed to the `claude-haiku-4-5` tier (1.00). The adjudication *share* falls as detection gets dearer, but adjudication *cost* rises, so a cheaper detection model does not simply refund its savings downstream. The `deepseek-v4-flash` arm was live on the first day only, which is why the cohort of Table 6 excludes that day.

Detection tier	Judge runs	Cost/run	Detection	Adjudication	Adjud. cost/run
`claude-haiku-4-5`	16.7%	1.00	9.1%	78.3%	1.00
`deepseek-v4-flash`	0.5%	1.38	24.5%	66.4%	1.17
`claude-sonnet-4-6`	68.7%	2.26	25.3%	60.5%	1.74
`claude-opus-5`	14.1%	4.34	55.4%	32.0%	1.76