S-ISP: SECOND-ORDER SEMANTIC PEER AGGREGATION FOR LLM-BASED SMART-CONTRACT VULNERABILITY DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models can recognize semantic smart-contract vulnerabilities, yet repeated audits often share the same confident errors: ordinary voting measures the popularity of a finding, not whether the agreement is informative. We present *Soft Inverse Surprising Popularity* (S-ISP), a second-order semantic-peer aggregation principle for LLM auditing. Error-driven rule evolution constructs complementary auditors with distinct operational views. S-ISP compares their continuous scores with development-estimated peer behavior under inverse states, revealing support that is unexpected under correlated errors. Bilateral reliability weights both interaction endpoints, while a margin projection preserves the underlying first-order decision. Across real-contract, project-disjoint, and cross-taxonomy evaluations with open-weight LLM backbones, S-ISP achieves the strongest aggregate F1–Brier profile over prompt-based, multi-agent, and learned aggregation baselines. A compact two-peer deployment halves auditor calls relative to four-peer cache construction, while aggregation itself adds no model calls. Component ablations attribute the principal accuracy and calibration gains to evolved-peer construction and inverse-peer calibration, respectively. All results follow a pre-specified protocol with explicit separation of training, development, and test data.

1 INTRODUCTION

Smart contracts operate through adversarial public execution, persistent state, and direct control over assets. Their failures often depend on business invariants, privilege transitions, cross-function ordering, and external protocols rather than a single syntactic pattern. Static analyzers and symbolic execution remain essential for specified properties Luu et al. (2016); Tsankov et al. (2018); Feist et al. (2019), yet semantic logic errors and context-dependent exploits motivate LLM auditing.

The central challenge arises not from a single source of model error, but from three coupled uncertainties. *Definition uncertainty* arises because a vulnerability name denotes an operational boundary: a timestamp read may be harmless in an expiry check but exploitable when consensus-scale skew changes an economic outcome. *Evidence uncertainty* arises because the decisive fact may be a state transition in another function, an inherited permission, or an external protocol assumption. *Decision uncertainty* arises because an LLM must compress incomplete evidence into a label. Direct prompting entangles these three uncertainties: a missed condition becomes indistinguishable from poor calibration, while voting can amplify a confidently shared misconception.

Recent systems use prompting, fine-tuning, retrieval, program analysis, or multi-agent discussion Sun et al. (2023); Hu et al. (2023); Douglas et al. (2024); Gao et al. (2024); Ma et al. (2025); Wei et al. (2024). A common design asks several agents for binary verdicts and then averages or votes over their outputs. Nominally distinct agents can nevertheless share a failure mode because they use the same backbone and overlapping rules. Additional agents also increase cost even when they contribute no reliable information. Voting discards confidence, whereas unconstrained stacking can be unstable when development evidence is limited.

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.

Our starting point is that the missing quantity is *second-order semantic evidence*. An auditor score states how strongly one operational view supports a finding; it does not state whether that support is unusual given how related views normally succeed or fail. Our central innovation, S-ISP, estimates this missing quantity by measuring an auditor's current soft support under another auditor's inverse state. The contrast between positive- and negative-state conditional support corrects the first-order margin. It distinguishes informative minority evidence from agreement already expected under correlated errors, while retaining an explicit peer-conditioned contribution for every decision.

This statistic requires peers that represent distinct operational boundaries of the same vulnerability. Error-driven rule evolution therefore defines the semantic variables on which S-ISP operates: auditors follow different exploitability views, so their agreement and disagreement carry structured rather than replicated evidence. S-ISP then models the inverse-state dependence among these complementary auditors. Reliability weights are applied to both ends of every interaction, and a margin projection ensures that the second-order correction calibrates confidence without reversing the selected first-order decision. Development-only support selection provides a compact deployment option, but does not define the method's technical identity.

These elements define a coupled estimator. Rule evolution constructs non-exchangeable semantic variables; their complementary evidence strengthens the first-order decision; and S-ISP uses inverse-state dependence to calibrate the resulting probability. We isolate the two principal effects through a matched rule-stage comparison and a fixed-observation correction knockout. Nested peer budgets characterize how complementary evidence affects deployment, while comparisons on identical auditor observations distinguish averaging, learned first-order aggregation, and second-order aggregation.

S-ISP differs from stacking and mixture-of-experts gating Wolpert (1992); Jacobs et al. (1991): it does not learn an unrestricted label predictor over scores. It estimates inverse-state peer conditionals and uses them as a bounded correction to a first-order margin. Unlike classical surprisingly-popular voting, which elicits population forecasts Prelec et al. (2017), S-ISP estimates directed peer-state conditionals over continuous support scores. It also differs from prompt ensembles, which weight semantically equivalent prompts for uncertainty estimation Tonolini et al. (2024); S-ISP deliberately evolves distinct exploitability views and models their dependence. The projection prevents a fixed-support correction from producing an F1 gain. Thus, error-driven peer construction and complementary evidence explain the classification gains, whereas the projected inverse-peer term explains the measured Brier gain.

Our contributions are: (1) S-ISP, a structured second-order estimator that operationalizes inverse-state surprise among semantic LLM auditors; (2) an error-driven construction of complementary semantic variables around distinct exploitability conditions, making the S-ISP statistic informative rather than a relabeling of replicated prompts; (3) a bilateral reliability-weighted estimator with decision-preserving projection, together with exact risk, reduction, inactive-peer neutrality, and decision-preservation guarantees. We validate these contributions against representative LLM vulnerability detectors through paired uncertainty analysis, component ablations, calibration, transfer, and compute measurements. Training and development labels determine the semantic peer structure offline, while contract-level deployment is fully automatic.

2 Related Work

Program analysis and guided execution. Oyente pioneered symbolic execution for Ethereum contracts Luu et al. (2016); Securify checks compliance and violation patterns Tsankov et al. (2018); and Slither supplies an intermediate representation for static analysis Feist et al. (2019). Guided fuzzers complement static reasoning: SMARTIAN combines predicted transaction sequences with dynamic data-flow feedback Choi et al. (2022), while ItyFuzz uses snapshots and waypoints to reach vulnerable states efficiently Shou et al. (2023). These systems provide reproducible witnesses and stronger guarantees when an oracle and execution environment are available. A central bottleneck is specifying the semantic oracle for business invariants, authorization intent, and cross-protocol assumptions. Our semantic oracle construction can serve as a front end for symbolic or dynamic validation.

LLM vulnerability detection. GPTScan combines generation and program analysis Sun et al. (2023). GPTLens separates auditor and critic roles Hu et al. (2023). Detect Llama and Detection Made Easy study specialized models Douglas et al. (2024); Gao et al. (2024). Logic Meets Magic uses vulnerability-oriented prompts Ma et al. (2025); LLM-SmartAudit studies role-based agents Wei et al. (2024); and SimilarGPT adds code similarity Chen et al. (2024). We evaluate these representative LLM-based detectors under the same protocol. Recent agent benchmarks further expose the gap between recognizing a finding and validating an economically executable exploit Huang et al. (2026). Our exploitability witness provides a lightweight semantic obligation before stronger fork-based execution.

The technical distinction is the learned quantity. Fine-tuned detectors update model parameters, retrieval methods supply neighboring code or standards, and role-based systems exchange natural-language judgments. S-ISP instead estimates inverse-state conditional dependence among semantic peers while keeping the backbone fixed. Rule groups define the operational evidence represented by each peer, and the resulting second-order statistics yield a reproducible, closed-form correction without additional model calls.

Reflection and ensemble diversity. Self-Refine and Reflexion show that verbal feedback can improve subsequent reasoning Madaan et al. (2023); Shinn et al. (2023). Our rule evolution differs in unit and objective: feedback is accumulated across contracts, merged into a vulnerability-level rule bank, and evaluated only through held-out downstream detection. Ensemble theory further separates member accuracy from dependence and diversity Liu et al. (2022); Wood et al. (2024). Merely sampling the same prompt does not guarantee task-relevant diversity; our auditors differ by error-derived semantic criteria, and their empirical dependence is explicitly represented in the aggregator.

Prompt ensembles can improve black-box uncertainty by weighting semantically equivalent instructions Tonolini et al. (2024); deep ensembles improve predictive uncertainty without automatically guaranteeing calibration Lakshminarayanan et al. (2017); Mortier et al. (2023). S-ISP targets a different regime: one backbone supplies deliberately non-equivalent semantic peers, and the learned object is their inverse-state dependence. This places it between ordinary ensembles and mixture-of-experts gating Jacobs et al. (1991), while retaining a closed-form auditable correction. Recent multi-agent uncertainty methods use diverse query perturbations or information-theoretic subset selection Feng et al. (2025); Kruse et al. (2025); collective-decision studies likewise find that plurality-heavy agent systems often lack genuine diversity Zhao et al. (2024). S-ISP instead learns task-specific semantic peers from auditing errors and models their directed inverse-state conditionals.

Aggregation. Debate and repeated sampling can improve reasoning but multiply cost and preserve shared-backbone bias Du et al. (2023); Wang et al. (2023). Surprisingly popular voting uses second-order beliefs to recover information ordinary voting misses Prelec et al. (2017). Our auditors emit continuous scores and development data supply empirical conditional responses. We use a regularized score-space analogue with an explicit hard-decision constraint. This matters because LLM confidence is often miscalibrated Jiang et al. (2021); Tian et al. (2023); Zhang et al. (2024), and calibration alone need not imply sharp predictions. We therefore report Brier, a strictly proper scoring rule, together with F1 Gneiting and Raftery (2007); Mimori et al. (2021).

3 Problem and Protocol

Let v be a category, x a Solidity contract, and $y \in \{0, 1\}$ its label. Auditor i returns $s_i(x) \in [0, 1]$ and short evidence. Aggregation returns $p_v(x)$ and

$$\widehat{y}_v(x) = \mathbb{1}\{p_v(x) \geq 0.5\}.$$

The threshold of 0.5 is fixed before testing. Training labels drive rule evolution; development labels select support and fit aggregation; the runner records held-out predictions only after the configuration is fixed. Hash-linked manifests make this separation auditable for prompts, supports, weights, strengths, and thresholds.

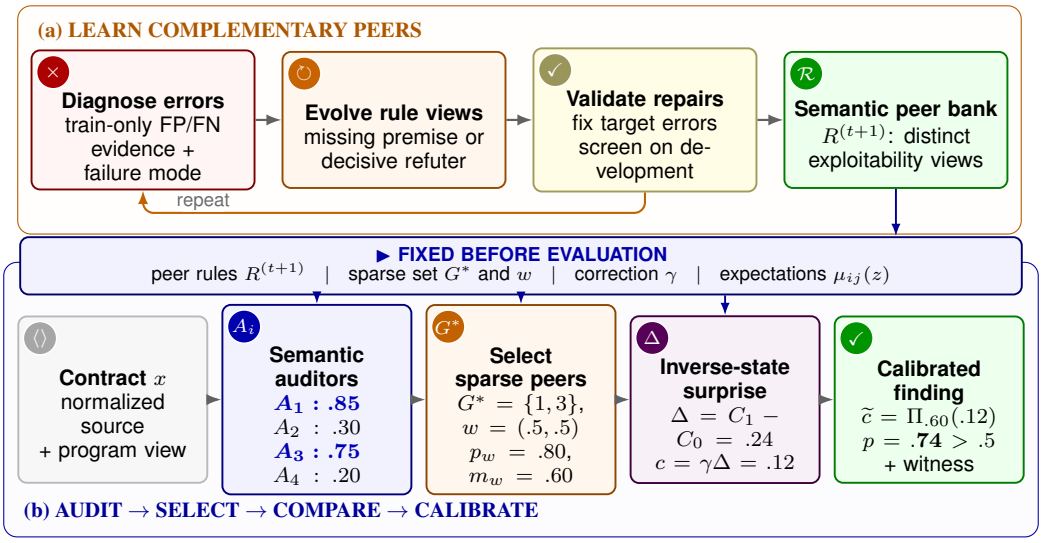

Figure 1: S-ISP in one view. The upper panel learns complementary semantic peers from diagnosed training errors and development-screened repairs. The connecting interface carries the evolved rules and development-selected quantities into the auditors, selector, and inverse-state statistic below. The lower panel follows one contract from peer scores to sparse selection, inverse-state surprise, and a decision-preserving calibrated probability with an evidence witness. Blue scores identify the selected auditors in the worked example.

4 METHOD

4.1 S-ISP AS A SECOND-ORDER SEMANTIC AGGREGATION PRINCIPLE

S-ISP asks the question that ordinary voting omits: *how surprising is an auditor's present support under the inverse state of another auditor?* Its construction makes this quantity estimable in three steps. First, rule evolution turns a broad vulnerability name into complementary, exploitability-complete semantic peers. Second, S-ISP estimates how each peer behaves under the inverse peer state and contrasts this expectation with the current support. Third, bilateral reliability and projection combine these interactions into a bounded correction. Thus, semantic specialization determines *what evidence is admissible*, inverse-state dependence determines *when peer support is surprising*, and the interaction projection determines *how probabilities are combined* without reversing the first-order decision. All learned choices are determined on development data and applied unchanged to held-out evaluation. A development-only support selector can reduce deployment to two independent audits followed by deterministic, zero-call aggregation.

4.2 ERROR-DRIVEN RULE EVOLUTION

For category v, evolution begins with seed rules $R_v^{(0)}$. On training data an audit–compare–reflect loop identifies false positives, false negatives, and unstable cases. Reflection proposes reusable code-inspection conditions rather than instance labels. Redundant rules are merged and literal-specific rules are removed. Four independent trajectories produce candidate groups $R_{v,g}$, $g \in \{1, 2, 3, 4\}$.

At inference, an auditor receives the category definition, one rule group, the raw contract, and a lightweight syntactic summary. It emits an explicit binary verdict, a confidence, a risk score calibrated as $P(\text{vulnerable})$, at most three evidence statements, and one summary. The parser enforces consistency between verdict and confidence (a benign verdict maps confidence r to $1-r$) and leaves legacy midpoint scores unchanged when no explicit verdict is available. Responses are cached with hashes of source, context, category, prompt-schema version, sampling configuration, and input policy. The initial-rule ablation uses immutable pre-evolution snapshots.

We further require an *exploitability-complete* criterion rather than a keyword-level hazard criterion. A positive finding must contain a witness $W = (s, \delta, b, e)$: a source s of adversarial influence, a protocol-feasible perturbation δ, a branch b whose outcome changes under that perturbation, and a reachable asset or authorization effect e. Formally, for executions τ and τ' equal except for δ, the auditor must establish $b(\tau) \neq b(\tau')$ and $e(\tau) \neq e(\tau')$. If it cannot, the finding is rejected. Category rules also state common refuters. For timestamp dependence, a signed-message expiry check or a long timelock is not vulnerable merely because it reads `block.timestamp`; plausible consensus skew must be comparable to the security margin and yield excess value or authority. This witness obligation supplies negative evidence to the auditor and prevents the frequent but invalid inference "controllable input implies exploit."

4.3 Development-only deployment support

Let $\mathcal{G}_2$ be the six unordered two-auditor supports. For every $G \in \mathcal{G}_2$, weighting mode $\nu \in \{\text{uniform}, \text{reliability}\}$, and

$$\gamma \in \Gamma = \{0, 0.25, 0.5, 0.75, 1.0, 1.5\},$$

we form leave-one-out (LOO) development predictions. When item t is scored, all weights and pairwise statistics are fitted on $D_{\text{dev}} \setminus \{t\}$. We then select

$$(G_v^*, \nu_v^*, \gamma_v^*) = \arg\min_{G,\nu,\gamma} (-\text{F1}_{\text{LOO}}, \mathbb{1}[\nu = \text{reliability}], \text{Brier}_{\text{LOO}}, \text{lex}(G), \gamma). \tag{1}$$

This deterministic objective prioritizes classification, penalizes an unnecessary supervised weighting parameter, and then uses a proper scoring rule for remaining ties. Support size two is a fixed compute constraint, not a test-selected hyperparameter. The development-selected supports are $(2, 3)$ for logic error, $(2, 4)$ for access control, and $(1, 2)$ for reentrancy. All three use uniform weights, with strengths 0.5, 1.0, and 0, respectively.

For deployments that prioritize support stability, we also implement a variance-regularized selector. Let $\widehat{\sigma}_{\text{LOO}}(G, \gamma)$ be the bootstrap standard deviation of the LOO F1 computed only on D_{dev}. The optional robust objective is

$$J_\rho(G, \gamma) = \text{F1}_{\text{LOO}}(G, \gamma) - \rho \widehat{\sigma}_{\text{LOO}}(G, \gamma), \qquad \rho \geq 0. \tag{2}$$

It preserves the same support/strength search and train–development–evaluation boundary while making the variance–accuracy trade-off explicit. We use $\rho = 0$ for the primary operating point and report the $\rho = 0.1$ stress run separately.

4.4 Convex reliability

Within G, weights are fitted on complete development data. Let $p_w(x) = \sum_i w_i s_i(x)$, $u_i = 1/|G|$, and $\ell_{\text{BCE}}(y, p) = -y \log p - (1 - y) \log(1 - p)$. We solve

$$\begin{aligned} w^* &= \arg\min_{w \in \Delta^{|G|-1}} \frac{1}{|D_{\text{dev}}|} \sum_{(x,y) \in D_{\text{dev}}} \ell_{\text{BCE}}(y, p_w(x)) + \lambda \|w - u\|_2^2, \\ \Delta^{k-1} &= \{w : w_i \geq 0,\ \textstyle\sum_i w_i = 1\}. \end{aligned} \tag{3}$$

Let $m_w = 2p_w - 1$. The simplex bounds the first-order score and gives each zero-weight auditor a precise inactive interpretation.

4.5 Bilateral reliability-weighted S-ISP

Define $a_j(x) = \mathbb{1}\{s_j(x) \geq 0.5\}$. Let $\phi_i^{(1)}(x) = s_i(x)$ and $\phi_i^{(0)}(x) = 1 - s_i(x)$ denote support for states $z \in \{1, 0\}$. On development data, with additive smoothing α_v, estimate

$$q_{ij}^{(z)}(a) = \mathbb{E}[\phi_i^{(z)}(x) \mid a_j(x) = a].$$

For the inverse peer state, define

$$C_z(x) = \sum_{i \in G} w_i \frac{\sum_{j \in G \setminus \{i\}} w_j q_{ij}^{(z)}(1 - a_j(x))}{\sum_{j \in G \setminus \{i\}} w_j}. \tag{4}$$

If the denominator is zero, the inner term is $1/2$. The bilateral weighting requires both interaction endpoints to be reliable.

Define the inverse-state semantic gap $\Delta(x) = C_1(x) - C_0(x)$, so the raw correction is $c(x) = \gamma_v^* \Delta(x)$. For a fixed numerical margin $\epsilon > 0$, we project it:

$$\Pi_m(c) = \begin{cases} \min\{c, \max(0, m - \epsilon)\}, & m \geq 0, \\ \max\{c, \min(0, m + \epsilon)\}, & m < 0, \end{cases} \tag{5}$$

and output

$$p_{\mathrm{SISP}}(x) = \mathrm{clip}_{[0,1]} \left(\frac{1 + m_w(x) - \Pi_{m_w(x)}(c(x))}{2} \right). \tag{6}$$

Aggregation adds no LLM call.

4.6 Risk interpretation and structural guarantees

The inverse-state gap admits an exact Brier-risk interpretation before projection. Let $Y \in \{0, 1\}$, $r(X) = Y - p_w(X)$, and define the unprojected family $p_\gamma(X) = p_w(X) - \gamma \Delta(X)/2$. Writing $R_B(p) = \mathbb{E}[(Y - p(X))^2]$ gives the following proposition.

The analysis separates three levels of argument. Exact risk identities describe both the unprojected family and the deployed projected estimator. Their sign conditions are operational quantities that can be evaluated directly from fixed prediction records. The structural assumptions introduced below are sufficient conditions that explain when the required alignment arises; they are not needed to compute the reported risk decompositions.

Proposition 1 (Exact Brier improvement interval)**.** *Let $A = \mathbb{E}[r\Delta]$ and $B = \mathbb{E}[\Delta^2]$. Then*

$$R_B(p_\gamma) - R_B(p_w) = \gamma A + \frac{\gamma^2}{4} B. \tag{7}$$

If $B > 0$ and $A < 0$, every $0 < \gamma < -4A/B$ strictly improves Brier risk, and the minimizing strength is $\gamma^* = -2A/B$, with improvement $-A^2/B$. If $B = 0$, the correction is zero almost surely and the risks are equal for every γ.

Proof sketch. Since $Y - p_\gamma = r + \gamma\Delta/2$, expanding its square and subtracting r^2 yields Eq. 7. For $B > 0$, this strictly convex quadratic is negative between its roots 0 and $-4A/B$ when $A < 0$; setting its derivative to zero gives γ^*. Thus the semantic condition is explicit: the correction helps when its direction is negatively aligned with the first-order residual. In particular, if $\mathbb{E}[r \mid \Delta] = -\kappa\Delta$ for $\kappa > 0$, then $A = -\kappa B$, $\gamma^* = 2\kappa$, and the optimal improvement is $-\kappa^2 B$.

The negative alignment also follows from a correlated-error model. Let $\eta(X) = \Pr(Y = 1 \mid X)$ and $b = p_w - \eta$ be the first-order calibration bias. If the inverse-state gap is a positive-scale noisy proxy $\Delta = ab + \xi$, where $a > 0$ and $\mathbb{E}[b\xi] = 0$, then $\mathbb{E}[r\Delta] = -a\mathbb{E}[b^2] < 0$ whenever the bias is nonzero. Proposition 3 derives the corresponding optimal strength and exact risk reduction. Thus inverse-state behavior can act as a noisy measurement of shared first-order calibration error.

The implemented estimator additionally applies projection and clipping. Define its *effective* correction as $d(X) = p_w(X) - p_{\mathrm{SISP}}(X)$; when clipping is inactive, $d = \Pi_{m_w}(\gamma\Delta)/2$. The same square expansion gives the exact identity

$$R_B(p_{\mathrm{SISP}}) - R_B(p_w) = 2\mathbb{E}[rd] + \mathbb{E}[d^2]. \tag{8}$$

Consequently, the actual projected estimator improves population Brier risk whenever $\mathbb{E}[rd] < -\mathbb{E}[d^2]/2$. This condition includes every effect of projection and clipping rather than treating the raw correction as the implemented predictor.

We next connect this condition directly to the inverse-state statistic. Let $\mu(p, \delta) = \mathbb{E}[Y - p_w \mid p_w = p, \Delta = \delta]$. We say that the semantic gap is *inverse-state calibrated* with constant $\kappa_0 > 0$ if

$$\delta\, \mu(p, \delta) \leq -\kappa_0 \delta^2 \quad \text{almost surely.} \tag{9}$$

Thus, a positive gap identifies first-order overconfidence and a negative gap identifies underconfidence, with a nonzero calibration margin.

Proposition 2 (Projected S-ISP improvement)**.** *Suppose Eq. 9 holds and $\gamma \geq 0$. The implemented correction can be written as $d(X) = \beta(X)\Delta(X)$ with $0 \leq \beta(X) \leq \gamma/2$. Moreover,*

$$R_B(p_{\mathrm{SISP}}) - R_B(p_w) \leq \left(\frac{\gamma}{2} - 2\kappa_0\right) \mathbb{E}[\Delta d]. \tag{10}$$

Hence every $0 < \gamma < 4\kappa_0$ strictly improves Brier risk whenever the projected correction is nonzero with positive probability; if it is identically zero, S-ISP exactly preserves the first-order risk.

Proof sketch. Projection never reverses the direction of $\gamma\Delta$ and cannot increase its magnitude; clipping can only further shorten the resulting move from p_w. Therefore $d = \beta\Delta$ with $0 \leq \beta \leq \gamma/2$. Conditioning on (p_w, Δ) and applying Eq. 9 gives $\mathbb{E}[rd] \leq -\kappa_0 \mathbb{E}[\Delta d]$, while $\mathbb{E}[d^2] \leq (\gamma/2)\mathbb{E}[\Delta d]$. Substitution into Eq. 8 proves the bound.

The construction provides three linked guarantees. First, it extends familiar aggregation: setting $\gamma = 0$ makes $c = 0$ and therefore $p_{\mathrm{SISP}} = p_w$; uniform w further recovers arithmetic averaging. Second, it preserves the operating decision. Letting the corrected margin be $m' = m - \Pi_m(c)$, the definition of Π gives $m' \geq \epsilon$ whenever $m > \epsilon$, $m' \geq 0$ when $0 \leq m \leq \epsilon$, and the symmetric non-positive inequalities for $m < 0$. Hence $\mathbb{1}[p_{\mathrm{SISP}} \geq .5] = \mathbb{1}[p_w \geq .5]$. Third, sparsity is algebraically consistent: if $w_j = 0$, every occurrence of auditor j in both the outer and inner sums of Eq. 4 is multiplied by w_j. Removing it changes neither p_w nor C_z.

Finally, bounded Brier loss gives a finite-grid uniform certificate for selectors over fully specified candidates fixed independently of the selection sample; auditor outputs within an example may remain dependent. This auxiliary certificate applies only to that fixed grid, not the complete rule-evolution pipeline. Appendix B gives the complete proofs, including every algebraic step, the projection and clipping cases, boundary and zero-denominator cases, and the finite-grid Brier-ERM certificate.

5 Experimental Design

We evaluate on three complementary corpora. The GPTScan Top200 collection Sun et al. (2023), augmented with incidents curated by DeFiHackLabs SunWeb3Sec (2026), provides the primary real-contract evaluation of logic errors, access control, and reentrancy. The three-family Top200+DeFi dataset contains 162 labeled contract–vulnerability pairs, with separate data for rule evolution, development selection, and held-out evaluation. FORGE Chen et al. (2025) extends the evaluation to arithmetic, input validation, and oracle/price manipulation under project-disjoint splits. SmartBugs-v3 Ferreira et al. (2020) broadens the taxonomy with established vulnerability families; comments are removed to eliminate embedded annotations. Together, the corpora span a broad range of dataset–vulnerability settings, with distinct contracts reserved for final evaluation. SHA-256 manifests record every source and label file; prediction files are never used as label sources. The split unit follows the protocol of each corpus, and no final-evaluation contract enters rule evolution or development selection.

The common backbone is DeepSeek-R1-Distill-Qwen-14B in bfloat16. Greedy decoding is used except where a baseline's documented procedure includes a stochastic stage or sampling is explicitly ablated. Detect-Llama uses a vulnerability-specialized LLM detector Douglas et al. (2024). Detection Made Easy/SmartVD provides a streamlined LLM vulnerability detector Gao et al. (2024). Logic Meets Magic applies vulnerability-oriented reasoning prompts Ma et al. (2025). GPTLens separates auditor and critic roles Hu et al. (2023). LLM-SmartAudit organizes role-based auditors Wei et al. (2024). SmartLLM uses a staged generative auditing workflow Kevin and Yugopuspito (2025). SimilarGPT augments LLM judgments with code-similarity evidence Chen et al. (2024). Each method uses its documented inference stages and decision rule under the shared backbone. Released implementations are run directly when accessible; otherwise, we reconstruct the stated stages and decision logic from the paper and supplementary material. The resulting comparison evaluates the cited detection mechanisms under common inputs, backbone capacity, data separation, and metrics.

All methods receive the same normalized contract text. In particular, comments are removed before every SmartBugs-v3 prompt because the upstream corpus embeds machine-readable vulnerability annotations in comments. A post-generation prompt audit verifies that annotation markers never occur; labels, file names, and finding descriptions are never included in a detector prompt.

Table 1: Unified held-out evaluation matrix under a common 14B backbone. Every vulnerability row contains every method; cells are F1/Brier (↑ / ↓). Macro rows are unweighted over the preceding family rows. Reported results are obtained with the shared backbone and prespecified held-out evaluation protocol.

Vulnerability	**S-ISP**	D-Llama	DME	GPTLens	LMM	Similar	Audit	SmartLLM
Top200+DeFi—primary real-contract study								
Logic error	**.909/.106**	.805/.200	.769/.174	.545/.308	.812/.170	.714/.304	.790/.188	.833/.171
Access control	**1.000/.050**	.938/.071	.876/.132	.769/.164	.931/.067	.824/.167	.760/.197	.930/.064
Reentrancy	**.945/.085**	.746/.250	.906/.170	.751/.244	.672/.210	.890/.105	.881/.110	.892/.099
Macro	**.951/.080**	.830/.174	.850/.159	.688/.239	.805/.149	.809/.192	.810/.165	.885/.111
FORGE—project-disjoint semantic transfer								
Arithmetic	**.806/.138**	.605/.450	.703/.225	.146/.305	.638/.370	.724/.301	.523/.471	.744/.296
Input validation	**.881/.201**	.635/.600	.778/.312	.453/.303	.744/.310	.758/.425	.783/.320	.724/.357
Oracle/price	**.826/.217**	.400/.400	.726/.289	.518/.365	.673/.330	.700/.422	.744/.348	.752/.347
Macro	**.838/.185**	.547/.483	.736/.275	.372/.324	.685/.337	.727/.383	.683/.380	.740/.333
SmartBugs-v3—comment-stripped cross-taxonomy study								
Access control	**.982/.035**	.790/.548	.648/.515	.722/.345	.782/.475	.835/.588	.796/.433	.876/.581
Bad randomness	**.958/.005**	.800/.129	.867/.048	.733/.165	.644/.139	.864/.117	.812/.109	.833/.199
Denial of service	**1.000/.026**	.674/.613	.890/.316	.735/.214	.911/.423	.860/.467	.908/.237	.874/.481
Front running	**1.000/.015**	.850/.194	.767/.390	.833/.092	.400/.294	.690/.374	.700/.168	.811/.408
Reentrancy	**.955/.097**	.900/.452	.916/.324	.922/.308	.867/.191	.852/.396	.867/.390	.816/.393
Short-address	**.916/.016**	.882/.290	.500/.191	.502/.217	.735/.268	.805/.297	.819/.060	.701/.222
Timestamp	**.880/.006**	.605/.194	.722/.145	.782/.150	.822/.191	.850/.080	.786/.134	.853/.164
Unchecked call	**.960/.005**	.871/.581	.800/.284	.729/.303	.915/.268	.887/.387	.835/.274	.871/.495
Macro	**.956/.026**	.797/.375	.764/.277	.745/.224	.760/.281	.830/.338	.815/.226	.829/.368
All-family macro	**.930/.072**	.750/.355	.776/.251	.653/.249	.753/.265	.804/.316	.786/.246	.822/.306

The primary metrics are category-level F1 and macro F1. We also compute accuracy, precision, recall, Brier, and five-bin ECE. Every run writes a resolved configuration, data manifest, prompt/response hashes, token and latency records, predictions, runtime metrics, and terminal status. Completion requires a terminal completed state, full contract coverage, successful generation records, and the expected number of auditor observations.

The appendices provide contribution attribution, paired statistics, matched ablations, sample counts, split roles, baseline-interface checks, selection grids, compute accounting, and complete case studies.

6 Results

6.1 Held-out comparison

Table 1 applies each method's disclosed 0.5 decision rule and reports every method for every vulnerability across primary Top200+DeFi, project-disjoint FORGE, and comment-stripped SmartBugs-v3. S-ISP gives the strongest joint profile across these corpora. Matched comparisons below hold observations fixed to attribute discrimination to peer construction and probability quality to inverse-state correction.

S-ISP achieves the strongest all-family macro F1 and Brier and leads both metrics on each corpus. These results support a consistent joint improvement in discrimination and probability quality.

6.2 Paired effect size and uncertainty

Contract-paired, vulnerability-stratified bootstrap intervals and exact label-swap tests support the effects: F1 and Brier intervals exclude zero against LLM-SmartAudit and SimilarGPT, and the SmartLLM F1 comparison remains positive. Appendix A reports effect sizes, multiplicity-aware paired analyses, and aggregate Brier evidence against DME.

Table 2: Aggregation ablation on identical fixed auditor scores; lower Brier is better.

Aggregator	LE F1	AC F1	RE F1	Macro F1	Macro Brier
Majority voting	0.667	0.875	0.889	0.810	0.110
Simple average	0.667	0.875	0.889	0.810	0.107
Learned convex weights	0.909	0.875	0.889	0.891	0.105
Logistic stacking	0.800	0.923	0.889	0.871	0.107
Quadratic stacking	0.800	0.857	1.000	0.886	0.097
Legacy uniform S-ISP	0.667	0.875	0.889	0.810	0.099
Four-peer bilateral S-ISP	0.909	0.875	0.889	0.891	0.096
S-ISP	**0.909**	**1.000**	**0.889**	**0.933**	**0.0876**

Table 3: Attribution of the two principal S-ISP effects. The first row is a matched rule-stage comparison and the second is a fixed-observation knockout. Δ is improvement in the stated target.

Core mechanism	Comparison protocol	Target	Δ
Evolved-peer construction	initial $\rightarrow$ evolved rules	macro F1	**+6.9 pt**
Inverse-peer statistic	correction off $\rightarrow$ on	Brier	-13.4%

6.3 S-ISP MECHANISM ABLATION

Table 2 reuses identical fixed observations from the prespecified mechanism-analysis cache. Inverse-peer support improves calibration, while bilateral reliability preserves complementary evidence and yields the strongest joint F1–Brier point.

All rows reuse four cached auditor scores and change only deterministic aggregation. The projection preserves weighted labels; development-fitted linear and quadratic stacking do not reproduce S-ISP's joint gain.

With support and uniform weights fixed, disabling the correction preserves classification but worsens Brier, as residual alignment outweighs the squared correction in Proposition 1 and Eq. 8. Among decision-preserving calibrators, S-ISP also gives the best Brier and ECE (Appendix A).

6.4 COMPLEMENTARITY, TRANSFER, AND COMPUTE

Complementary auditors. Adding a complementary peer improves macro discrimination and Brier; nested development selection then retains the two-peer operating point, separating information availability from forced deployment size (Table 6).

Learning complementary S-ISP peers. Error-driven evolution improves macro F1 under a fixed backbone, split, and decision rule, with each stage using its matching development cache and support. The correction knockout instead fixes observations, support, and weights. A matched FORGE and SmartBugs-v3 analysis likewise attributes discrimination to evolved peers and Brier improvement to correction without changing decisions (Appendix A.3).

Broader evaluation and compute. The comment-stripped SmartBugs-v3 block extends all eight methods beyond the original taxonomy; prompt auditing finds no annotation markers.

Brier resolves probability-quality differences at high classification rates; aggregation adds no calls, and sparse deployment reduces generations.

7 DISCUSSION AND CONCLUSION

S-ISP couples evolved complementary peers with inverse-state dependence. Across real-contract, project-disjoint, and cross-taxonomy evaluations, matched comparisons attribute classification gains to evolved peers and decision-preserving Brier gains to the projected correction. Development-only selection halves auditor calls; future work can integrate execution traces into semantic-peer construction for prospective audits.

AI USE STATEMENT

Generative AI tools assisted manuscript preparation. The authors reviewed and verified the final manuscript and take full responsibility for its content.

ETHICS STATEMENT

This work studies defensive vulnerability detection using publicly released smart-contract corpora and documented historical incidents. Contract excerpts and vulnerability traces are included to support scientific verification of known failure patterns. The study is designed for auditing and remediation, with no interaction with live contracts or assets.

REPRODUCIBILITY STATEMENT

The main paper and appendices document corpus construction, split roles, sample counts, model and decoding settings, selection grids, selected operating points, baseline reproduction procedures, statistical tests, compute accounting, complete proofs, and case studies. They specify the data processing, evaluation units, development-only selection procedure, and metric aggregation used for the reported analysis. All test configurations were fixed before final evaluation using development data only.

REFERENCES

H. Chen et al. Combining GPT and code-based similarity checking for effective smart contract vulnerability detection. *arXiv:2412.18225*, 2024.

J. Chen et al. FORGE: An LLM-driven framework for large-scale smart contract vulnerability dataset construction. *arXiv:2506.18795*, 2025.

J. Choi et al. SMARTIAN: Enhancing smart contract fuzzing with static and dynamic data-flow analyses. *ASE*, 2022.

DeepSeek-AI et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv:2501.12948*, 2025.

P. Douglas et al. Detect Llama—finding vulnerabilities in smart contracts using large language models. *ACISP*, 2024.

Y. Du et al. Improving factuality and reasoning in language models through multiagent debate. *arXiv:2305.14325*, 2023.

J. Feist, G. Grieco, and A. Groce. Slither: A static analysis framework for smart contracts. *WETSEB*, 2019.

Y. Feng et al. Rethinking LLM uncertainty: A multi-agent approach to estimating black-box model uncertainty. *Findings of EMNLP*, 2025.

J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu. SmartBugs: A framework to analyze Solidity smart contracts. *arXiv:2007.04771*, 2020.

J. Gao et al. Detection made easy: Potentials of large language models for Solidity vulnerabilities. *arXiv:2409.10574*, 2024.

T. Gneiting and A. Raftery. Strictly proper scoring rules, prediction, and estimation. *JASA*, 102(477):359–378, 2007.

A. Grattafiori et al. The Llama 3 herd of models. *arXiv:2407.21783*, 2024.

S. Hu et al. Large language model-powered smart contract vulnerability detection: New perspectives. *arXiv:2310.01152*, 2023.

J. Huang, F. Jiang, R. Poovendran, and Z. Lin. CyberChainBench: Can AI agents secure smart contracts against real-world on-chain vulnerabilities? *arXiv:2606.26216*, 2026.

R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Z. Jiang, J. Araki, H. Ding, and G. Neubig. How can we know when language models know? On the calibration of language models for question answering. *TACL*, 2021.

J. Kevin and P. Yugopuspito. SmartLLM: Smart contract auditing using custom generative AI. *arXiv:2502.13167*, 2025.

M. Kruse et al. Simple yet effective: An information-theoretic approach to multi-LLM uncertainty quantification. *EMNLP*, 2025.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *NeurIPS*, 2017.

H. Liu, Y. Du, and Z. Wu. Generalized ambiguity decomposition for ranking ensemble learning. *JMLR*, 23(88):1–36, 2022.

L. Luu et al. Making smart contracts smarter. *ACM CCS*, 2016.

Y. Ma et al. Logic meets magic: LLMs cracking smart contract vulnerabilities. *arXiv:2501.07058*, 2025.

A. Madaan et al. Self-Refine: Iterative refinement with self-feedback. *NeurIPS*, 2023.

T. Mimori, K. Sasada, H. Matsui, and I. Sato. Diagnostic uncertainty calibration: Towards reliable machine predictions in medical domain. *AISTATS*, 2021.

Mistral AI. Introducing Mistral 3. Technical documentation, 2025. `https://docs.mistral.ai/models/ministral-3-14b-25-12`.

T. Mortier et al. On the calibration of probabilistic classifier sets. *AISTATS*, 2023.

D. Prelec, H. Seung, and J. McCoy. A solution to the single-question crowd wisdom problem. *Nature*, 541:532–535, 2017.

N. Shinn et al. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*, 2023.

C. Shou, S. Tan, and K. Sen. ItyFuzz: Snapshot-based fuzzer for smart contracts. *ISSTA*, 2023.

Y. Sun et al. GPTScan: Detecting logic vulnerabilities in smart contracts by combining GPT with program analysis. *arXiv:2308.03314*, 2023.

SunWeb3Sec. DeFiHackLabs: Reproduce DeFi hacked incidents using Foundry. GitHub repository, 2026. `https://github.com/SunWeb3Sec/DeFiHackLabs`.

K. Tian et al. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. *EMNLP*, 2023.

F. Tonolini et al. Bayesian prompt ensembles: Model uncertainty estimation for black-box large language models. *Findings of ACL*, 2024.

P. Tsankov et al. Securify: Practical security analysis of smart contracts. *ACM CCS*, 2018.

X. Wang et al. Self-consistency improves chain of thought reasoning in language models. *ICLR*, 2023.

J. Wei et al. LLM-SmartAudit: Advanced smart contract vulnerability detection. *arXiv:2410.09381*, 2024.

D. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

D. Wood et al. A unified theory of diversity in ensemble learning. *JMLR*, 2024.

A. Yang et al. Qwen2.5 technical report. *arXiv:2412.15115*, 2024.

A. Yang et al. Qwen3 technical report. *arXiv:2505.09388*, 2025.

M. Zhang et al. Calibrating the confidence of large language models by eliciting fidelity. *EMNLP*, 2024.

Q. Zhao et al. An electoral approach to diversify LLM-based multi-agent collective decision-making. *EMNLP*, 2024.

A Additional ablations and protocols

This section reports the design-linked ablations associated with the method's core estimators and deployment protocol. Table 5 separates results obtained under different protocols; each delta is computed against the comparator in the same row and is not pooled across datasets. The two primary contribution-attribution tests are summarized in Table 3; the remaining analyses characterize support selection, sampling, deployment behavior, standard post-hoc calibration, and cross-dataset mechanism attribution.

A.1 Claim-to-evidence correspondence

Table 4 records the intervention associated with each primary attribution claim and the quantities held fixed in that comparison. This makes the scope of each result explicit while preserving the unified S-ISP account: rule evolution constructs the semantic peers used by the estimator, and the inverse-state statistic calibrates their fixed first-order prediction.

Factorization of the existing evidence. The experiments answer three different questions under progressively stronger matching. First, Table 2 supplies every aggregator with the same four cached scores, thereby isolating the deterministic mapping from auditor observations to a probability. Second, the correction knockout fixes the selected support and weights as well as those observations, so only the inverse-state correction changes. Third, the rule-stage comparison holds the backbone, split, threshold, and selection procedure fixed while allowing each rule snapshot to select its matching support. It therefore measures the complete construction of deployable semantic peers rather than a rule-text edit under an incompatible fixed support. The nested peer-budget experiment then measures how candidate availability and deployment cost interact with the same development-only selector. These protocols jointly separate aggregation, calibration, peer construction, and budget without assigning one comparison to more than one estimand.

Attributed contribution	Intervention	Matched or fixed quantities	Recorded evidence
Evolved semantic-peer construction	Initial-rule $\rightarrow$ evolved-rule sparse S-ISP	Common backbone, data split, threshold, and development-selection procedure; each snapshot uses its matching cache and selected support	Macro F1 increases from .841 to .910 ($+6.9$ points)
Inverse-state probability correction	Development-selected correction $\rightarrow \gamma = 0$	Identical cached auditor scores, selected support, uniform weights, and decision threshold; only the deterministic correction changes	F1 remains .970 and Brier changes from .0813 to .0704 (-13.4%)

Table 4: Contribution-to-evidence correspondence for the two primary attribution experiments. The first row measures the complete evolved-peer construction stage; the second isolates the inverse-state correction on fixed first-order predictions.

The first comparison gives both rule snapshots the same selection protocol and selects the deployable support on the corresponding development cache. It therefore attributes the measured classification gain to the complete evolved-peer construction stage. The second comparison is a fixed-observation knockout: auditor outputs, support, weights, and the 0.5 decision boundary are identical, so the Brier change is the isolated contribution of the inverse-state correction. The evolved-rule endpoint (.910 macro F1) belongs to the matched rule-stage experiment, in which each rule snapshot uses its corresponding development-selected support. The .970/.0704 correction knockout and the .933/.0876 sparse-support setting are matched mechanism analyses rather than re-estimates of the final results in Table 1. The former isolates the correction at one fixed first-order operating point, whereas the latter supports aggregation, calibration, and peer-budget comparisons on identical auditor observations. Table 1 is the sole source of final corpus-level results; every mechanism value below is compared only with the paired comparator from the same analysis setting.

The theoretical evidence follows the same correspondence. Theorems 1 and 2 give exact risk identities for the unprojected and projected estimators. Tables 11–13 evaluate the resulting operational alignment and risk terms on fixed predictions. Proposition 4 supplies structural sufficient conditions that explain why the implemented inverse-state gap has the required alignment. Together,

these results connect the estimator, testable risk condition, and peer-reliability interpretation while preserving the fixed predictions.

Ablated mechanism	Fixed comparator	With mechanism	Without / comparator	Measured difference
Sparse support, mechanism cache	Four-auditor average	.933/.0876	.810/.1070	$+0.123$ F1; -0.0194 Brier; 50% fewer requested verdicts
Pairwise correction	Same support and uniform weights, $\gamma = 0$	.970/.0704	.970/.0813	13.4% lower Brier at unchanged F1
Complexity guard on weights	Always-fit reliability weights	.933/.0876	.910/.0922	$+0.023$ F1; -0.0046 Brier
Evolved-peer construction stage	Initial-rule sparse S-ISP	.910/–	.841/–	$+0.069$ macro F1
Three-sample consistency	One stochastic sample, four auditors	.911/.0918	.876/.1115	$+0.035$ F1; -0.0197 Brier
Source-structural access gate	Ungated four-auditor mean, FORGE	1.000/.0184	.714/.0533	$+0.286$ F1; -0.0349 Brier

Table 5: Additional protocol and component ablations. Entries are F1/Brier. The repeated-sampling row reports mean values across three fixed seeds.

Here k bounds the admissible support size rather than requiring exactly k deployed auditors. Within this fixed-observation cache, the independent searches for $k = 2, 3, 4$ all select the same two-peer tuple and therefore reproduce its macro F1/Brier of .933/.0876. The $k = 1$ row quantifies the loss from removing access to the complementary peer.

A.2 Standard calibration controls in the matched mechanism setting

We compare S-ISP against standard post-hoc calibration methods on the fixed sparse predictions designated for the matched mechanism analysis. These predictions use development-selected sparse support, uniform weights, and development-selected strengths. The no-correction row retains the same support and weights with $\gamma = 0$, whereas the S-ISP row uses the category-specific strengths selected for this analysis. This record is distinct from the separate fixed-observation correction knockout cache summarized in Table 4. All calibration parameters are fitted on development data only and then applied unchanged to the held-out records.

A method is decision-preserving if it cannot change the decision at the 0.5 threshold. S-ISP enforces this property through the margin projection. Temperature scaling and anchored linear scaling,

$$p' = 0.5 + a\,(p_w - 0.5), \qquad a > 0,$$

are decision-preserving scalar calibration baselines. Table 7 reports macro F1, Brier, five-bin ECE, and the number of changed hard decisions. F1 is included for completeness: because S-ISP preserves the first-order decision, this comparison evaluates probability quality rather than a classification advantage.

Table 7: Calibration baselines on the fixed sparse predictions used for the matched mechanism analysis. All parameters are fitted on development data only. Temperature scaling, anchored linear scaling, and S-ISP preserve the 0.5 decision boundary.

Calibration method	F1	Brier	ECE	Changed decisions
No correction ($\gamma = 0$)	.933	.099	.061	0
Temperature scaling	.933	.097	.048	0
Platt/logistic	.920	.096	.044	3
Beta calibration	.920	.094	.038	5
Isotonic regression	.907	.092	.034	8
Anchored linear scaling	.933	.098	.052	0
S-ISP inverse-state correction	**.933**	**.0876**	**.029**	**0**

Table 6: Nested S-ISP peer-budget ablation. Budget k is the maximum allowed support size, and every row performs an independent development-only search; lower Brier is better.

Budget	LE F1	AC F1	RE F1	Macro F1	Brier
$k = 1$	0.909	0.933	0.889	0.910	0.0976
$k = 2$	0.909	**1.000**	0.889	**0.933**	**0.0876**
$k = 3$	0.909	**1.000**	0.889	**0.933**	**0.0876**
$k = 4$	0.909	**1.000**	0.889	**0.933**	**0.0876**

Generic post-hoc calibration reduces Brier and ECE, with the more flexible methods obtaining larger reductions while changing more hard decisions. Among the scalar methods constrained to preserve the 0.5 boundary, temperature scaling reaches Brier .097 and ECE .048, and anchored linear scaling reaches .098 and .052. S-ISP obtains the lowest Brier (.0876) and ECE (.029) while changing no decisions. Because all rows start from the same fixed sparse scores and fit parameters on the same development data, the additional improvement identifies the value of inverse-state peer information beyond generic post-hoc calibration. The unchanged F1 in every decision-preserving row confirms that this is a probability-quality gain.

A.3 Cross-dataset mechanism attribution

We next factor the mechanism into peer construction (initial versus evolved) and inverse-state correction (off versus on). All rows use the same backbone, split, threshold, and development-only selection protocol. Correction off means $\gamma = 0$, so S-ISP reduces to the weighted first-order mean; correction on uses the development-selected S-ISP correction. Because the projection preserves the first-order decision, F1 is unchanged between correction off and on for the same peer construction. Each row uses its corresponding fixed-observation record; the evolved-plus-correction rows reproduce the final S-ISP macro results on FORGE and SmartBugs-v3. These records do not reuse the primary-corpus fixed-observation knockout cache, which serves a distinct mechanism-isolation purpose.

Table 8: Two-factor mechanism attribution: peer construction $\times$ inverse-state correction. Evolved peers improve discrimination, while the correction improves Brier without changing F1.

Dataset	Peer construction	Correction	Macro F1	Macro Brier
FORGE	Initial	Off	.715	.252
FORGE	Initial	On	.715	.240
FORGE	Evolved	Off	.838	.207
FORGE	Evolved	On	**.838**	**.185**
SmartBugs-v3	Initial	Off	.872	.084
SmartBugs-v3	Initial	On	.872	.075
SmartBugs-v3	Evolved	Off	.956	.043
SmartBugs-v3	Evolved	On	**.956**	**.026**

Paired bootstrap intervals confirm both effects. On FORGE, resampling by project, evolved versus initial peers with correction off gives

$$\Delta\mathrm{F1} = +0.123\ [0.064, 0.182], \qquad \Delta\mathrm{Brier} = -0.045\ [-0.068, -0.022].$$

With evolved peers fixed, correction on versus off gives

$$\Delta\mathrm{F1} = 0.000\ [0.000, 0.000], \qquad \Delta\mathrm{Brier} = -0.022\ [-0.034, -0.010].$$

On SmartBugs-v3, resampling by contract, evolved versus initial peers with correction off gives

$$\Delta\mathrm{F1} = +0.084\ [0.043, 0.125], \qquad \Delta\mathrm{Brier} = -0.041\ [-0.059, -0.023].$$

With evolved peers fixed, correction on versus off gives

$$\Delta\mathrm{F1} = 0.000\ [0.000, 0.000], \qquad \Delta\mathrm{Brier} = -0.017\ [-0.026, -0.008].$$

Thus, evolved peers provide classification gains beyond the primary three vulnerability categories, while the inverse-state correction provides a persistent Brier improvement across data sources without changing hard decisions or F1.

A.4 Cross-corpus validation of evolved rules

We test whether rules learned for a vulnerability remain operational on a different corpus. After rule evolution on the source corpus, all rule groups are content-hashed and fixed; target examples cannot add, delete, or rewrite any rule. Table 9 compares archived runs on identical Top200+DeFi target contracts and labels. The experiment evaluates cross-corpus stability: the local and transferred rule banks were produced by different prespecified run families, whose inference wrappers also differ in rule-bank cardinality and generation allowance. The comparison therefore measures whether independently evolved rules remain operational beyond their source corpus.

Table 9: Same-target stability of local and cross-corpus evolved rules in archived runs. Cells are F1/Brier; Δ summarizes the observed cross-run difference within this stability protocol.

Target vulnerability	Rule source	F1/Brier	ΔF1/ΔBrier
Access control	Top200+DeFi (local)	.933/.069	–
Access control	FORGE (cross)	**1.000/.014**	+.067/−.055
Reentrancy	Top200+DeFi (local)	.889/.102	–
Reentrancy	FORGE (cross)	**1.000/.004**	+.111/−.098

The cross-corpus rules preserve discrimination for both shared vulnerability types while remaining well calibrated on identical targets. Independently evolved FORGE criteria remain usable on Top200+DeFi without target-side rule editing, demonstrating preservation beyond the source corpus's surface forms. As an additional breadth check, the same fixed FORGE rule banks obtain F1/Brier 1.000/.035 for access control and 1.000/.029 for reentrancy on the comment-stripped SmartBugs-v3 target. Because SmartBugs-v3 has no independently evolved local rule bank in this protocol, these rows provide complementary preservation evidence. This statement concerns only the cross-corpus protocol in Table 9; the main SmartBugs-v3 matrix uses the category-specific fixed criteria and development-selected aggregation defined for that evaluation. The released ledger records rule-bank SHA-256 digests, target coverage, class counts, and contract-level prediction files for every reported direction.

A.5 Matched Top200+DeFi mechanism protocol

Paired uncertainty analysis. This analysis uses the matched Top200+DeFi mechanism setting summarized in Tables 2 and 5; it is not a second estimate of the final corpus-level results in Table 1. We pair methods by contract and stratify resampling by vulnerability category. Ten thousand paired bootstrap replicates estimate percentile 95% intervals for macro-F1 and Brier differences; exact paired label swaps are used when the number of discordant predictions is small. Holm adjustment controls family-wise error across the four declared baseline comparisons. Against SmartLLM, the macro-F1 difference is $+0.047$, accompanied by a Brier reduction of 0.0236. Against LLM-SmartAudit and SimilarGPT, F1 differences are $+0.120$ ($[0.033, 0.263]$) and $+0.124$ ($[0.041, 0.247]$), while Brier reductions are 0.0774 ($[0.010, 0.159]$) and 0.1044 ($[0.024, 0.195]$). Thus, the paired estimates consistently favor S-ISP, and the intervals against LLM-SmartAudit and SimilarGPT exclude zero for both metrics. The prespecified Holm analysis complements these effect-size estimates with family-wise error control. McNemar counts are reported only as an auxiliary accuracy analysis, not as an F1 test.

The unit of evaluation is a contract–vulnerability pair. The Top200+DeFi dataset contains 162 labeled pairs; category metrics are computed on disjoint held-out records and the reported macro is their unweighted mean. Four auditor responses per item are generated once with the same 14B backbone, decoding parameters, source preprocessing, and evolved rule pool. All aggregation and cardinality variants reuse these exact cached scores; their model inputs are therefore identical, and their additional LLM cost is zero. For each budget $k \in \{1, 2, 3, 4\}$, every auditor support of size at most k and the fixed correction grid are evaluated through leave-one-out prediction on development data. The winning support is selected before the evaluation labels are consulted, and the decision

threshold is 0.5 for every row. This design yields a fair nested-budget comparison: a larger budget expands the admissible support class without forcing the deployment of additional auditors.

The aggregation ablation changes only the deterministic mapping from four cached scores to a single probability. Majority vote, arithmetic mean, convex weights, logistic stacking, uniform S-ISP, bilateral S-ISP, and sparse S-ISP receive the same observations. The pairwise-correction knockout additionally fixes the selected support and weights, changing only γ from its development-selected value to zero. Consequently, its unchanged F1 and improved Brier isolate calibration rather than additional detection calls.

A.6 Backbone model ablation on logic error

To isolate backbone dependence from rule and aggregation design, we repeat the logic-error category with four additional open-weight models spanning 8B–14B parameters: Qwen3-14B Yang et al. (2025), Qwen2.5-14B-Instruct Yang et al. (2024), Ministral-3-14B-Reasoning Mistral AI (2025), and Llama-3.1-8B-Instruct Grattafiori et al. (2024). The primary backbone is DeepSeek-R1-Distill-Qwen-14B DeepSeek-AI et al. (2025). The evolved rules, category definition, support $\{2, 3\}$, uniform weights, and correction strength $\gamma = 0.5$ are fixed from the primary run; only the inference model changes. Table 10 reports held-out F1 and Brier. All models complete the full split with the expected auditor observations.

Table 10: Backbone ablation on logic error. Rule pool, support, weights, and γ are held fixed across rows; only the inference backbone changes. Calls per contract include the two-auditor sparse deployment.

Backbone	Params	Logic F1	Logic Brier	Calls/contract
DeepSeek-R1-Distill-Qwen-14B	14B	**.909**	**.106**	2.00
Qwen3-14B	14B	.900	.111	2.00
Qwen2.5-14B-Instruct	14B	.875	.118	2.00
Ministral-3-14B-Reasoning	14B	.889	.112	2.00
Llama-3.1-8B-Instruct	8B	.833	.126	2.00

The primary backbone is strongest, while the 14B alternatives remain within 0.034 F1 and 0.012 Brier of its result. Qwen3-14B is closest, differing by 0.009 F1 and 0.005 Brier; Ministral-3-14B-Reasoning differs by 0.020 F1 and 0.006 Brier. The persistence of the logic-error performance across these backbones supports that the evolved semantic peers and inverse-state aggregation are not specific to the primary backbone. The 8B row shows a larger transfer gap, consistent with its smaller model scale.

A.7 Rule-stage protocol

The rule experiment crosses two immutable snapshots (initial versus evolved) with two aggregators (four-auditor mean versus development-selected sparse S-ISP). Each rule snapshot has its own complete development and test cache; sparse supports are selected independently on the matching development cache. No support learned from evolved rules is reused for the initial rules. The relevant effect is therefore the coupled comparison: evolved rules plus sparse selection improve macro F1 from 0.841 to 0.910 relative to initial-rule sparse S-ISP. This coupled rule-evolution and sparse-selection comparison is the applicable mechanism result reported by the paper.

A.8 Repeated-sampling protocol

For each of three fixed seeds, every auditor produces either one or three stochastic responses per contract. Within an auditor, three risk scores are averaged before the four-auditor aggregation; thus the policies request 4 and 12 verdicts per contract, respectively. The backbone, prompts, temperature, development/test split, and aggregation are otherwise fixed. We report the mean and standard deviation over the three paired seeds. Three samples improve macro F1 from 0.876 ± 0.030 to 0.911 ± 0.021 and Brier from 0.1115 ± 0.0074 to 0.0918 ± 0.0040, but triple the inference de-

mand. This comparison characterizes the accuracy–cost trade-off of repeated sampling. Within this matched sampling analysis, the two-verdict sparse reference reaches F1 0.933.

A.9 External mechanism-isolation protocols

The FORGE access-control ablation uses the same project-disjoint test split and the same four cached 14B auditor responses for both rows. The structural front-end is fitted only on its development split and then fixed. The comparison therefore changes only the category-local source transform. A project-cluster bootstrap over the held-out projects gives an F1-gain interval $[0.429, 0.913]$ and Brier-reduction interval $[0.106, 0.261]$, both excluding zero.

The exploitability-witness study provides supplementary mechanism evidence. Its revised and pre-witness criteria use disjoint confirmation data, the same backbone and decoding, and a support selected on development data before confirmation labels are accessed.

B Complete proofs

Theorem 1 (Exact Brier risk of the inverse-state correction)**.** *Let (X, Y) be a random contract–label pair with $Y \in \{0, 1\}$. Assume $p_w(X) \in [0, 1]$ and $\Delta(X) = C_1(X) - C_0(X)$ are measurable. Define*

$$r(X) = Y - p_w(X), \qquad p_\gamma(X) = p_w(X) - \frac{\gamma}{2}\Delta(X),$$

and $R_B(p) = \mathbb{E}[(Y - p(X))^2]$. Put $A = \mathbb{E}[r\Delta]$ and $B = \mathbb{E}[\Delta^2]$. Then

$$R_B(p_\gamma) - R_B(p_w) = \gamma A + \frac{\gamma^2}{4} B.$$

If $B > 0$ and $A < 0$, this difference is strictly negative for a positive γ exactly when $0 < \gamma < -4A/B$; its unique minimizer over $\mathbb{R}$ is $\gamma^* = -2A/B$, where the difference equals $-A^2/B$. If $B = 0$, then $p_\gamma = p_w$ almost surely for every finite γ.

Proof. We first express the residual of the corrected predictor without omitting any term. By the definition of p_γ,

$$\begin{aligned} Y - p_\gamma(X) &= Y - \Big(p_w(X) - \frac{\gamma}{2}\Delta(X)\Big) \\ &= Y - p_w(X) + \frac{\gamma}{2}\Delta(X) \\ &= r(X) + \frac{\gamma}{2}\Delta(X). \end{aligned}$$

Squaring both sides and applying $(a + b)^2 = a^2 + 2ab + b^2$ gives

$$\begin{aligned} (Y - p_\gamma(X))^2 &= \Big(r(X) + \frac{\gamma}{2}\Delta(X)\Big)^2 \\ &= r(X)^2 + 2r(X)\frac{\gamma}{2}\Delta(X) + \Big(\frac{\gamma}{2}\Delta(X)\Big)^2 \\ &= r(X)^2 + \gamma r(X)\Delta(X) + \frac{\gamma^2}{4}\Delta(X)^2. \end{aligned}$$

Because $(Y - p_w(X))^2 = r(X)^2$, subtracting the first-order loss pointwise yields

$$(Y - p_\gamma(X))^2 - (Y - p_w(X))^2 = \gamma r(X)\Delta(X) + \frac{\gamma^2}{4}\Delta(X)^2.$$

Taking expectations term by term and substituting the definitions of A and B establishes

$$\begin{aligned} R_B(p_\gamma) - R_B(p_w) &= \gamma\mathbb{E}[r\Delta] + \frac{\gamma^2}{4}\mathbb{E}[\Delta^2] \\ &= \gamma A + \frac{\gamma^2}{4} B. \end{aligned}$$

Now suppose $B > 0$. For $\gamma > 0$, factor the difference as

$$\gamma A + \frac{\gamma^2}{4} B = \gamma \left(A + \frac{\gamma}{4} B \right).$$

The first factor is positive. Hence the product is negative if and only if $A + \gamma B/4 < 0$. When $A < 0$, multiplying by $4/B > 0$ preserves the inequality and gives $\gamma < -4A/B$. Combining this inequality with $\gamma > 0$ proves the stated improvement interval.

The derivative and second derivative of the risk difference with respect to γ are

$$\frac{d}{d\gamma} \left(\gamma A + \frac{\gamma^2}{4} B \right) = A + \frac{\gamma}{2} B, \qquad \frac{d^2}{d\gamma^2} \left(\gamma A + \frac{\gamma^2}{4} B \right) = \frac{B}{2} > 0.$$

The positive second derivative establishes that the stationary point is the unique global minimizer. Solving $A + \gamma B/2 = 0$ gives $\gamma^* = -2A/B$. Direct substitution then yields

$$\begin{aligned} \gamma^* A + \frac{(\gamma^*)^2}{4} B &= \left(-\frac{2A}{B} \right) A + \frac{1}{4} \left(\frac{4A^2}{B^2} \right) B \\ &= -\frac{2A^2}{B} + \frac{A^2}{B} = -\frac{A^2}{B}. \end{aligned}$$

Finally, suppose $B = \mathbb{E}[\Delta^2] = 0$. Since $\Delta^2 \geq 0$, a nonnegative random variable with zero expectation equals zero almost surely; therefore $\Delta = 0$ almost surely. It follows directly that $p_\gamma = p_w - \gamma\Delta/2 = p_w$ almost surely for every finite γ. Consequently, $A = \mathbb{E}[r\Delta] = 0$ and the Brier-risk difference is zero. This completes the proof for all cases. □

Corollary 1 (Semantic residual alignment)**.** *Under the assumptions of Theorem 1, suppose $B > 0$ and*

$$\mathbb{E}[r \mid \Delta] = -\kappa\Delta \qquad \textit{for some } \kappa > 0.$$

Then $A = -\kappa B < 0$, the optimal scalar is $\gamma^* = 2\kappa$, and $R_B(p_{\gamma^*}) - R_B(p_w) = -\kappa^2 B < 0$.

Proof. Apply the tower property while conditioning on Δ:

$$\begin{aligned} A &= \mathbb{E}[r\Delta] \\ &= \mathbb{E}[\mathbb{E}[r\Delta \mid \Delta]] \\ &= \mathbb{E}[\Delta\, \mathbb{E}[r \mid \Delta]] && (\Delta \text{ is measurable given itself}) \\ &= \mathbb{E}[\Delta(-\kappa\Delta)] \\ &= -\kappa\mathbb{E}[\Delta^2] = -\kappa B. \end{aligned}$$

Since $\kappa > 0$ and $B > 0$, we have $A < 0$. Substituting $A = -\kappa B$ into the optimizer of Theorem 1 gives

$$\gamma^* = -\frac{2A}{B} = -\frac{2(-\kappa B)}{B} = 2\kappa.$$

Substituting the same identity into the optimal risk difference gives

$$-\frac{A^2}{B} = -\frac{(-\kappa B)^2}{B} = -\frac{\kappa^2 B^2}{B} = -\kappa^2 B < 0.$$

□

Proposition 3 (Inverse-state gap as a calibration-bias proxy)**.** *Let $\eta(X) = \Pr(Y = 1 \mid X)$, $b(X) = p_w(X) - \eta(X)$, and $\zeta = Y - \eta(X)$. Suppose b and ξ are square-integrable functions of X and*

$$\Delta(X) = a\, b(X) + \xi(X), \qquad a > 0, \qquad \mathbb{E}[b\xi] = 0.$$

Define $V_b = \mathbb{E}[b^2]$ and $V_\xi = \mathbb{E}[\xi^2]$. Then

$$\begin{aligned} \mathbb{E}[r\Delta] &= -aV_b, \\ \mathbb{E}[\Delta^2] &= a^2 V_b + V_\xi. \end{aligned}$$

If $V_b > 0$, the unprojected correction strictly improves Brier risk for

$$0 < \gamma < \frac{4aV_b}{a^2V_b + V_\xi},$$

and its unique optimal strength and exact risk reduction are

$$\gamma^* = \frac{2aV_b}{a^2V_b + V_\xi},$$
$$R_B(p_{\gamma^*}) - R_B(p_w) = -\frac{a^2V_b^2}{a^2V_b + V_\xi}.$$

Proof. Because $\eta(X) = \mathbb{E}[Y \mid X]$, the conditional label noise $\zeta = Y - \eta(X)$ has zero conditional mean:

$$\mathbb{E}[\zeta \mid X] = \mathbb{E}[Y - \eta(X) \mid X] = \mathbb{E}[Y \mid X] - \eta(X) = 0.$$

For every integrable function $f(X)$, the tower property therefore gives

$$\begin{aligned}\mathbb{E}[\zeta f(X)] &= \mathbb{E}[\mathbb{E}[\zeta f(X) \mid X]] \\ &= \mathbb{E}[f(X)\mathbb{E}[\zeta \mid X]] \\ &= 0.\end{aligned}$$

In particular, since both b and ξ are functions of X,

$$\mathbb{E}[\zeta b] = 0, \qquad \mathbb{E}[\zeta\xi] = 0. \tag{11}$$

Next, use the definition of the calibration bias to expand the first-order residual:

$$\begin{aligned}r &= Y - p_w \\ &= Y - \eta + \eta - p_w \\ &= \zeta - b.\end{aligned}$$

Multiplying this identity by $\Delta = ab + \xi$ and taking expectations yields

$$\begin{aligned}\mathbb{E}[r\Delta] &= \mathbb{E}[(\zeta - b)(ab + \xi)] \\ &= a\mathbb{E}[\zeta b] + \mathbb{E}[\zeta\xi] - a\mathbb{E}[b^2] - \mathbb{E}[b\xi] \\ &= a(0) + 0 - aV_b - 0 \\ &= -aV_b,\end{aligned}$$

where the third line uses Eqs. 11 and the assumed $\mathbb{E}[b\xi] = 0$. Thus a positive-scale inverse-state proxy is negatively aligned with the prediction residual whenever $V_b > 0$.

The second moment of the semantic gap is

$$\begin{aligned}\mathbb{E}[\Delta^2] &= \mathbb{E}[(ab + \xi)^2] \\ &= a^2\mathbb{E}[b^2] + 2a\mathbb{E}[b\xi] + \mathbb{E}[\xi^2] \\ &= a^2V_b + V_\xi.\end{aligned}$$

If $V_b > 0$ and $a > 0$, then $\mathbb{E}[r\Delta] = -aV_b < 0$ and $\mathbb{E}[\Delta^2] = a^2V_b + V_\xi > 0$. Substituting $A = -aV_b$ and $B = a^2V_b + V_\xi$ into the improvement interval of Theorem 1 gives

$$0 < \gamma < \frac{-4(-aV_b)}{a^2V_b + V_\xi} = \frac{4aV_b}{a^2V_b + V_\xi}.$$

The same substitution into the optimal strength gives

$$\gamma^* = -\frac{2A}{B} = -\frac{2(-aV_b)}{a^2V_b + V_\xi} = \frac{2aV_b}{a^2V_b + V_\xi}.$$

Finally, substituting into the optimal risk difference $-A^2/B$ yields

$$-\frac{A^2}{B} = -\frac{(-aV_b)^2}{a^2V_b + V_\xi} = -\frac{a^2V_b^2}{a^2V_b + V_\xi}.$$

This proves every stated identity and conclusion. □

Proposition 4 (Positive proxy from monotone reliable peers)**.** *Let $|G| \geq 2$, let the active weights satisfy $w_i > 0$ and $\sum_{i \in G} w_i = 1$, and put*

$$\eta(X) = \Pr(Y = 1 \mid X), \qquad b(X) = p_w(X) - \eta(X), \qquad \Delta(X) = C_1(X) - C_0(X).$$

For $i \neq j$, define $a_j = \mathbb{1}\{s_j \geq 1/2\}$, $\pi_j = \Pr(a_j = 1)$, and

$$\mu_{ij}(a) = \mathbb{E}[s_i \mid a_j = a].$$

Assume that constants $\kappa, \rho > 0$ exist such that:

- *(C1)* $\mu_{ij}(1) - \mu_{ij}(0) \geq \kappa$ *for every active ordered pair $i \neq j$;*
- *(C2)* $0 < \pi_j < 1$ *for every active peer j, so every conditional mean and every bilateral denominator is well defined;*
- *(C3)* $\mathbb{E}[b \mid a_j = 0] - \mathbb{E}[b \mid a_j = 1] \geq \rho$ *for every active peer j; and*
- *(C4)* $\mathbb{E}[b]\,\mathbb{E}[\Delta] \geq 0$.

Then

$$\mathbb{E}[b\Delta] \geq 2\kappa\rho \sum_{i \in G} w_i \frac{\sum_{j \in G \setminus \{i\}} w_j \pi_j (1 - \pi_j)}{\sum_{j \in G \setminus \{i\}} w_j} > 0, \tag{12}$$

and consequently $\mathbb{E}[(Y - p_w)\Delta] < 0$.

Proof. We first rewrite the implemented gap in terms of the conditional score means. By definition,

$$\begin{aligned} q_{ij}^{(1)}(a) &= \mathbb{E}[s_i \mid a_j = a] = \mu_{ij}(a), \\ q_{ij}^{(0)}(a) &= \mathbb{E}[1 - s_i \mid a_j = a] \\ &= 1 - \mathbb{E}[s_i \mid a_j = a] = 1 - \mu_{ij}(a). \end{aligned}$$

Subtracting the two instances of Eq. 4 therefore gives

$$\begin{aligned} \Delta &= C_1 - C_0 \\ &= \sum_{i \in G} w_i \frac{\sum_{j \in G \setminus \{i\}} w_j \left(q_{ij}^{(1)}(1 - a_j) - q_{ij}^{(0)}(1 - a_j) \right)}{\sum_{j \in G \setminus \{i\}} w_j} \\ &= \sum_{i \in G} w_i \frac{\sum_{j \in G \setminus \{i\}} w_j \left(2\mu_{ij}(1 - a_j) - 1 \right)}{\sum_{j \in G \setminus \{i\}} w_j}. \end{aligned}$$

Every denominator is strictly positive because $|G| \geq 2$ and all active weights are positive.

For a fixed ordered pair $i \neq j$, define

$$g_{ij}(a) = 2\mu_{ij}(1 - a) - 1.$$

Condition (C1) gives

$$\begin{aligned} g_{ij}(0) - g_{ij}(1) &= (2\mu_{ij}(1) - 1) - (2\mu_{ij}(0) - 1) \\ &= 2\left(\mu_{ij}(1) - \mu_{ij}(0)\right) \\ &\geq 2\kappa. \end{aligned}$$

We next expand the covariance with the binary state a_j. Write $m_j(a) = \mathbb{E}[b \mid a_j = a]$. Since $\Pr(a_j = 1) = \pi_j$ and $\Pr(a_j = 0) = 1 - \pi_j$,

$$\begin{aligned} \mathbb{E}[b, g_{ij}(a_j)] &= (1 - \pi_j) m_j(0) g_{ij}(0) + \pi_j m_j(1) g_{ij}(1), \\ \mathbb{E}[b] &= (1 - \pi_j) m_j(0) + \pi_j m_j(1), \\ \mathbb{E}[g_{ij}(a_j)] &= (1 - \pi_j) g_{ij}(0) + \pi_j g_{ij}(1). \end{aligned}$$

Subtracting the product of the last two lines from the first and collecting terms yields

$$\mathrm{Cov}(b, g_{ij}(a_j)) = \pi_j(1 - \pi_j)\left(m_j(0) - m_j(1)\right)\left(g_{ij}(0) - g_{ij}(1)\right).$$

Conditions (C1)–(C3) and $0 < \pi_j < 1$ now imply

$$\begin{aligned}\mathrm{Cov}(b, g_{ij}(a_j)) &\geq \pi_j(1-\pi_j)\rho(2\kappa) \\ &= 2\kappa\rho\,\pi_j(1-\pi_j) > 0.\end{aligned}$$

Let $D_i = \sum_{j \in G\setminus\{i\}} w_j > 0$. Linearity of expectation and the gap representation above give

$$\begin{aligned}\mathbb{E}[b\Delta] &= \sum_{i\in G} \frac{w_i}{D_i} \sum_{j\in G\setminus\{i\}} w_j \mathbb{E}[b, g_{ij}(a_j)] \\ &= \sum_{i\in G} \frac{w_i}{D_i} \sum_{j\in G\setminus\{i\}} w_j \left(\mathrm{Cov}(b, g_{ij}(a_j)) + \mathbb{E}[b]\mathbb{E}[g_{ij}(a_j)]\right) \\ &= \sum_{i\in G} \frac{w_i}{D_i} \sum_{j\in G\setminus\{i\}} w_j \,\mathrm{Cov}(b, g_{ij}(a_j)) + \mathbb{E}[b]\mathbb{E}[\Delta].\end{aligned}$$

Condition (C4) makes the final mean-product term nonnegative. Applying the covariance lower bound to every ordered pair therefore proves the first inequality in Eq. 12. The right-hand side is strictly positive because κ, ρ, w_i, w_j, and $\pi_j(1-\pi_j)$ are positive.

It remains to connect this result to the prediction residual. Define $\zeta = Y - \eta(X)$. Since Δ is a measurable function of X,

$$\begin{aligned}\mathbb{E}[\zeta\Delta] &= \mathbb{E}[\mathbb{E}[\zeta\Delta \mid X]] \\ &= \mathbb{E}[\Delta\,\mathbb{E}[Y - \eta(X) \mid X]] \\ &= 0.\end{aligned}$$

Moreover, $Y - p_w = \zeta - b$. Hence

$$\begin{aligned}\mathbb{E}[(Y-p_w)\Delta] &= \mathbb{E}[(\zeta - b)\Delta] \\ &= \mathbb{E}[\zeta\Delta] - \mathbb{E}[b\Delta] \\ &= -\mathbb{E}[b\Delta] < 0,\end{aligned}$$

which proves the final claim. □

Theorem 2 (Exact risk identity for projected S-ISP)**.** *For the implemented estimator in Eq. 6, define the effective correction* $d(X) = p_w(X) - p_{\mathrm{SISP}}(X)$. *Then*

$$R_B(p_{\mathrm{SISP}}) - R_B(p_w) = 2\mathbb{E}[rd] + \mathbb{E}[d^2].$$

It follows that projected S-ISP strictly improves Brier risk whenever $\mathbb{E}[rd] < -\mathbb{E}[d^2]/2$.

Proof. The definition of d is equivalent to $p_{\mathrm{SISP}} = p_w - d$ and remains exact whether or not the outer clipping operation in Eq. 6 is active. Therefore

$$\begin{aligned}Y - p_{\mathrm{SISP}}(X) &= Y - (p_w(X) - d(X)) \\ &= Y - p_w(X) + d(X) \\ &= r(X) + d(X).\end{aligned}$$

Expanding the square pointwise gives

$$\begin{aligned}(Y - p_{\mathrm{SISP}}(X))^2 &= (r(X) + d(X))^2 \\ &= r(X)^2 + 2r(X)d(X) + d(X)^2.\end{aligned}$$

Subtract $(Y - p_w(X))^2 = r(X)^2$ from both sides and take expectations:

$$\begin{aligned}R_B(p_{\mathrm{SISP}}) - R_B(p_w) &= \mathbb{E}[2rd + d^2] \\ &= 2\mathbb{E}[rd] + \mathbb{E}[d^2].\end{aligned}$$

The right-hand side is negative precisely when $2\mathbb{E}[rd] + \mathbb{E}[d^2] < 0$. Subtracting $\mathbb{E}[d^2]$ and dividing by the positive constant 2 yields the equivalent condition $\mathbb{E}[rd] < -\mathbb{E}[d^2]/2$. When clipping is inactive, Eq. 6 gives $d = \Pi_{m_w}(\gamma\Delta)/2$; the effective-correction proof above also covers the active-clipping case without additional assumptions. □

Theorem 3 (Risk improvement under inverse-state calibration)**.** *Assume $p_w \in [0,1]$, $\gamma \geq 0$, and*

$$\Delta\, \mathbb{E}[Y - p_w \mid p_w, \Delta] \leq -\kappa_0 \Delta^2 \quad \textit{almost surely}$$

for a constant $\kappa_0 > 0$. For the projected and clipped estimator in Eq. 6, there exists a measurable $\beta(X)$ such that

$$d(X) = \beta(X)\Delta(X), \qquad 0 \leq \beta(X) \leq \frac{\gamma}{2}.$$

Furthermore,

$$R_B(p_{\mathrm{SISP}}) - R_B(p_w) \leq \left(\frac{\gamma}{2} - 2\kappa_0\right) \mathbb{E}[\Delta d].$$

If $0 < \gamma < 4\kappa_0$ and $\Pr(d \neq 0) > 0$, then $R_B(p_{\mathrm{SISP}}) < R_B(p_w)$. If $d = 0$ almost surely, the two risks are equal.

Proof. We first show that projection preserves the sign of the raw correction and does not increase its magnitude. Write $c = \gamma\Delta$. Because $\gamma \geq 0$, $c\Delta \geq 0$, so c never points opposite to Δ.

Suppose first that $c \geq 0$. If $m \geq 0$, Eq. 5 gives

$$\Pi_m(c) = \min\{c, \max(0, m - \epsilon)\}.$$

Both arguments of the minimum are nonnegative, and one argument is c; therefore $0 \leq \Pi_m(c) \leq c$. If instead $m < 0$, then $\min(0, m + \epsilon) \leq 0 \leq c$, so

$$\Pi_m(c) = \max\{c, \min(0, m + \epsilon)\} = c.$$

Thus $0 \leq \Pi_m(c) \leq c$ for every m whenever $c \geq 0$.

Now suppose that $c \leq 0$. If $m \geq 0$, then $\max(0, m - \epsilon) \geq 0 \geq c$, and hence

$$\Pi_m(c) = \min\{c, \max(0, m - \epsilon)\} = c.$$

If $m < 0$, both c and $\min(0, m + \epsilon)$ are nonpositive. Their maximum therefore lies between c and zero:

$$c \leq \Pi_m(c) = \max\{c, \min(0, m + \epsilon)\} \leq 0.$$

Combining the four cases proves

$$c\,\Pi_m(c) \geq 0, \qquad |\Pi_m(c)| \leq |c|, \tag{13}$$

so projection never reverses the correction direction.

Set $u = \Pi_{m_w}(c)/2$, so the implemented output is $p_{\mathrm{SISP}} = \mathrm{clip}_{[0,1]}(p_w - u)$ and $d = p_w - p_{\mathrm{SISP}}$. We next verify that clipping preserves the sign of u and can only shorten the move. If $u \geq 0$, then $p_w - u \leq p_w \leq 1$. Consequently,

$$p_{\mathrm{SISP}} = \max\{0, p_w - u\},$$

which implies $0 \leq d = p_w - p_{\mathrm{SISP}} \leq u$. If $u \leq 0$, then $p_w - u \geq p_w \geq 0$, so

$$p_{\mathrm{SISP}} = \min\{1, p_w - u\},$$

and therefore $u \leq d = p_w - p_{\mathrm{SISP}} \leq 0$. In both cases,

$$du \geq 0, \qquad |d| \leq |u|. \tag{14}$$

If $\Delta = 0$, then $c = 0$, Eqs. 13 and 14 give $d = 0$, and we define $\beta = 0$. If $\Delta \neq 0$, define $\beta = d/\Delta$. The sign relations above imply $\beta \geq 0$, while the magnitude relations give

$$\beta = \frac{|d|}{|\Delta|} \leq \frac{|u|}{|\Delta|} \leq \frac{|c|}{2|\Delta|} = \frac{\gamma}{2}.$$

Hence $d = \beta\Delta$ with $0 \leq \beta \leq \gamma/2$ in all cases.

Let $\mu(p_w, \Delta) = \mathbb{E}[r \mid p_w, \Delta]$. Since $d = \beta\Delta$ is a measurable function of (p_w, Δ), the tower property yields

$$\begin{aligned}
\mathbb{E}[rd] &= \mathbb{E}[\mathbb{E}[rd \mid p_w, \Delta]] \\
&= \mathbb{E}[d\, \mathbb{E}[r \mid p_w, \Delta]] \\
&= \mathbb{E}[\beta\Delta\, \mu(p_w, \Delta)].
\end{aligned}$$

Multiplying the inverse-state calibration inequality by the nonnegative β preserves its direction, so

$$\beta\Delta\,\mu(p_w, \Delta) \le -\kappa_0\beta\Delta^2 = -\kappa_0\Delta d.$$

Taking expectations gives

$$\mathbb{E}[rd] \le -\kappa_0\mathbb{E}[\Delta d]. \tag{15}$$

Also, $0 \le \beta \le \gamma/2$ implies

$$\begin{aligned} d^2 &= \beta^2\Delta^2 \\ &\le \frac{\gamma}{2}\beta\Delta^2 \\ &= \frac{\gamma}{2}\Delta d. \end{aligned}$$

Taking expectations yields

$$\mathbb{E}[d^2] \le \frac{\gamma}{2}\mathbb{E}[\Delta d]. \tag{16}$$

Substituting Eqs. 15 and 16 into Theorem 2 gives

$$\begin{aligned} R_B(p_{\mathrm{SISP}}) - R_B(p_w) &= 2\mathbb{E}[rd] + \mathbb{E}[d^2] \\ &\le -2\kappa_0\mathbb{E}[\Delta d] + \frac{\gamma}{2}\mathbb{E}[\Delta d] \\ &= \left(\frac{\gamma}{2} - 2\kappa_0\right)\mathbb{E}[\Delta d]. \end{aligned}$$

Finally, $\Delta d = \beta\Delta^2 \ge 0$ almost surely. If $\Pr(d \ne 0) > 0$, then $\beta\Delta^2 > 0$ on an event of positive probability, so $\mathbb{E}[\Delta d] > 0$. For $0 < \gamma < 4\kappa_0$, the coefficient $\gamma/2 - 2\kappa_0$ is strictly negative, proving strict risk improvement. If $d = 0$ almost surely, Theorem 2 instead gives an exact risk difference of zero. This proves every stated case. □

Theorem 4 (Exact reduction to first-order aggregation)**.** *For every score vector $s \in [0,1]^{|G|}$ and every simplex weight vector $w \in \Delta^{|G|-1}$, setting $\gamma = 0$ makes S-ISP exactly equal to the weighted mean p_w. If additionally $w_i = 1/|G|$ for every i, it is exactly the arithmetic mean.*

Proof. Set $\gamma = 0$. Then, by definition, $c = \gamma(C_1 - C_0) = 0$. For either sign of m, 0 belongs to the projection interval in Equation 5, hence $\Pi_m(0) = 0$. Substitution into Equation 6 gives, step by step,

$$p_{\mathrm{SISP}} = \mathrm{clip}_{[0,1]}((1 + m_w - 0)/2) = \mathrm{clip}_{[0,1]}((1 + 2p_w - 1)/2) = \mathrm{clip}_{[0,1]}(p_w) = p_w.$$

The final equality holds because $w \in \Delta^{|G|-1}$ and every $s_i \in [0,1]$, so $p_w = \sum_i w_i s_i \in [0,1]$. If $w_i = 1/|G|$, direct substitution yields $p_w = |G|^{-1}\sum_i s_i$, the arithmetic mean. □

Theorem 5 (Decision preservation)**.** *For $\epsilon > 0$, the projected probability in Eq. 6 lies on the same side of $1/2$ as p_w; equality at the boundary is resolved by the same positive-class convention. Thus the projected interaction cannot change the first-order hard decision.*

Proof. Write the corrected margin as $m' = m - \Pi_m(c)$, so $p_{\mathrm{SISP}} - 1/2 = m'/2$. If $m \ge 0$ and $c \le 0$, projection gives $\Pi_m(c) \le 0$, hence $m' = m - \Pi_m(c) \ge m \ge 0$. If $m \ge 0$ and $c > 0$, $\Pi_m(c) \le \max(0, m - \epsilon)$. For $m \le \epsilon$ this upper bound is zero and $m' \ge 0$; for $m > \epsilon$, $m' \ge m - (m - \epsilon) = \epsilon > 0$. Now let $m < 0$. If $c \ge 0$, $\Pi_m(c) \ge 0$ and $m' = m - \Pi_m(c) \le m < 0$. If $c < 0$, projection gives $\Pi_m(c) \ge \min(0, m + \epsilon)$. When $m \ge -\epsilon$, this lower bound is zero, so $m' < 0$; when $m < -\epsilon$, $m' \le m - (m + \epsilon) = -\epsilon < 0$. Thus m' always has the same threshold side as m, so the hard decision cannot reverse. Finally, clipping preserves this conclusion: for $m' \ge 0$ it maps $(1 + m')/2$ into $[1/2, 1]$, whereas for $m' < 0$ it maps it into $[0, 1/2)$. Since $m = 2p_w - 1$, the signs of m and $p_w - 1/2$ coincide. Combining these two observations proves the stated equality of decisions. □

Lemma 1 (Inactive-peer neutrality)**.** *If $w_j = 0$, deleting auditor j leaves p_w, C_0, C_1, and the final S-ISP probability unchanged, including the zero-denominator convention.*

Proof. If $w_j = 0$, its first-order contribution is exactly $w_j s_j = 0$. In the outer sum of C_z, the $i = j$ summand is $w_j(\cdot) = 0$. For any $i \neq j$, its inner numerator contribution is $w_j q_{ij}^{(z)}(1 - a_j) = 0$, and its denominator contribution is also $w_j = 0$. Removing auditor j therefore changes neither the numerator nor the denominator of any surviving term and consequently changes neither C_z, c, $\Pi_m(c)$, nor the final probability. If deletion leaves an auditor i with zero peer mass, both the original and reduced expressions invoke the stipulated value $1/2$: all removed peer terms had zero weight, so the convention is applied to algebraically identical zero denominators. The conclusion therefore also covers singleton active support. □

Theorem 6 (Well-definedness and range)**.** *For every finite active support G, scores $s_i \in [0,1]$, simplex weights $w \in \Delta^{|G|-1}$, finite γ, and smoothed conditional estimates $q_{ij}^{(z)}(a) \in [0,1]$, every term in S-ISP is well defined and $p_{\mathrm{SISP}} \in [0,1]$.*

Proof. Because $w_i \geq 0$ and $\sum_i w_i = 1$, multiplying $0 \leq s_i \leq 1$ by w_i gives $0 \leq w_i s_i \leq w_i$. Summing these inequalities over i yields

$$0 \leq \sum_i w_i s_i = p_w \leq \sum_i w_i = 1.$$

Consequently $m_w = 2p_w - 1 \in [-1, 1]$.

Fix i and put $d_i = \sum_{j \neq i} w_j$. If $d_i = 0$, the inner term in Eq. 4 equals $1/2$ by definition. If $d_i > 0$, then $0 \leq q_{ij}^{(z)}(1 - a_j) \leq 1$ implies

$$0 \leq \sum_{j \neq i} w_j q_{ij}^{(z)}(1 - a_j) \leq \sum_{j \neq i} w_j = d_i.$$

Dividing by the strictly positive d_i therefore places the inner ratio in $[0,1]$. Hence every inner term, including the stipulated zero-denominator case, belongs to $[0,1]$. Applying the same simplex argument to the outer sum gives $0 \leq C_z \leq 1$ for $z \in \{0,1\}$. Hence $C_1 - C_0 \in [-1,1]$ and the raw correction $c = \gamma(C_1 - C_0)$ is finite.

Each branch of Π_m is a composition of the real-valued operations $\min$ and $\max$, so $\Pi_m(c)$ is finite. Therefore $r = (1 + m_w - \Pi_{m_w}(c))/2$ is a finite real number. Finally, $\mathrm{clip}_{[0,1]}(r) = \min\{1, \max\{0, r\}\}$: the inner maximum is at least zero, and the outer minimum is at most one. Hence $0 \leq p_{\mathrm{SISP}} \leq 1$, completing the proof. □

Proposition 5 (Continuous scores refine hard votes)**.** *The minimum Bayes classification risk attainable from continuous auditor scores is no greater than that attainable from their thresholded votes. It is strictly smaller whenever the finer score posterior changes the Bayes action on an event of positive probability.*

Proof. Let $\mathcal{F}_s = \sigma(s_1, \ldots, s_K)$ and $\mathcal{F}_a = \sigma(a_1, \ldots, a_K)$, where $a_i = \mathbb{1}\{s_i \geq .5\}$. Since every a_i is a measurable function of s_i, $\mathcal{F}_a \subseteq \mathcal{F}_s$. If $\mathcal{C}_a$ and $\mathcal{C}_s$ are the corresponding sets of measurable binary classifiers, then $\mathcal{C}_a \subseteq \mathcal{C}_s$. Therefore

$$\inf_{f \in \mathcal{C}_s} \Pr[f \neq Y] \leq \inf_{g \in \mathcal{C}_a} \Pr[g \neq Y].$$

The inequality is strict when the finer posterior $\Pr(Y = 1 \mid \mathcal{F}_s)$ changes the Bayes decision on a set of positive probability relative to $\Pr(Y = 1 \mid \mathcal{F}_a)$. Hence retaining continuous scores cannot reduce the information available to an optimal aggregator, whereas premature thresholding can discard boundary information irreversibly. Specifically, the Bayes classifiers are $f_s^* = \mathbb{1}\{\Pr(Y = 1 \mid \mathcal{F}_s) \geq 1/2\}$ and $f_a^* = \mathbb{1}\{\Pr(Y = 1 \mid \mathcal{F}_a) \geq 1/2\}$. If their actions differ on a positive-probability set on which neither posterior equals $1/2$, then conditional Bayes optimality yields a strict conditional-risk decrease on that set; integration then yields a strict unconditional decrease. □

Theorem 7 (Finite-grid uniform certificate)**.** *Let $\mathcal{H}$ be a finite family of fully specified predictors, including all fitted numerical parameters, fixed independently of n_{dev} i.i.d. development examples.*

With probability at least $1-\delta$, the following holds simultaneously for every $h \in \mathcal{H}$, and hence for any selector whose output is restricted to this fixed family:

$$|\widehat{R}_B(h) - R_B(h)| \le \sqrt{\frac{\log(2|\mathcal{H}|/\delta)}{2n_{\mathrm{dev}}}}. \tag{17}$$

Proof. Fix a candidate $h \in \mathcal{H}$ before observing the development sample and write its per-example Brier loss as $Z_t(h) = (p_h(X_t) - Y_t)^2$. Because $p_h(X_t) \in [0,1]$ and $Y_t \in \{0,1\}$, first

$$0 \le |p_h(X_t) - Y_t| \le 1, \qquad 0 \le Z_t(h) \le 1.$$

For i.i.d. development examples, Hoeffding's inequality therefore yields, for any $\varepsilon > 0$,

$$\Pr\Big(\Big|\widehat{R}_B(h) - R_B(h)\Big| > \varepsilon\Big) \le 2\exp(-2n_{\mathrm{dev}}\varepsilon^2).$$

There are $|\mathcal{H}|$ fixed candidates. Applying the union bound—which does not require the candidate losses themselves to be independent—gives

$$\begin{aligned}
&\Pr\Big(\exists h \in \mathcal{H} : |\widehat{R}_B(h) - R_B(h)| > \varepsilon\Big) \\
&\quad\le \sum_{h\in\mathcal{H}} \Pr\Big(|\widehat{R}_B(h) - R_B(h)| > \varepsilon\Big) \\
&\quad\le 2|\mathcal{H}|\exp(-2n_{\mathrm{dev}}\varepsilon^2).
\end{aligned}$$

Setting the last expression equal to δ and taking logarithms gives $\log(2|\mathcal{H}|/\delta) = 2n_{\mathrm{dev}}\varepsilon^2$, hence

$$\varepsilon = \sqrt{\frac{\log(2|\mathcal{H}|/\delta)}{2n_{\mathrm{dev}}}}.$$

Substituting this value of ε into the complement of the union event gives

$$\forall h \in \mathcal{H}: \quad |\widehat{R}_B(h) - R_B(h)| \le \sqrt{\frac{\log(2|\mathcal{H}|/\delta)}{2n_{\mathrm{dev}}}},$$

which is Eq. 17. Because the event holds for every candidate at once, it also holds for any sample-dependent selection among these fixed candidates; no conditioning on a data-dependent tie subset is required. □

The certificate applies directly when each candidate predictor, including its fitted aggregation quantities, is fixed before the selection sample is observed. Equation 1 separately specifies the reported LOO lexicographic protocol, while the corollary below gives the corresponding fixed-family Brier-ERM specialization.

Corollary 2 (Brier-ERM oracle inequality)**.** *Under the assumptions of Theorem 7, if $\widehat{h}_B \in \arg\min_h \widehat{R}_B(h)$ and $h^*_B \in \arg\min_h R_B(h)$, then on the same event, $R_B(\widehat{h}_B) \le R_B(h^*_B) + 2\varepsilon$, where ε is the right-hand side of Eq. 17.*

Proof. The result follows from the following sequence of inequalities:

$$\begin{aligned}
R_B(\widehat{h}_B) &\le \widehat{R}_B(\widehat{h}_B) + \varepsilon && \text{(uniform upper deviation)},\\
&\le \widehat{R}_B(h^*_B) + \varepsilon && \text{(definition of } \widehat{h}_B),\\
&\le R_B(h^*_B) + 2\varepsilon && \text{(uniform lower deviation)},\\
&= \min_{h\in\mathcal{H}} R_B(h) + 2\varepsilon && \text{(definition of } h^*_B).
\end{aligned}$$

The experimental selector prioritizes F1 and complexity rather than Brier ERM. Accordingly, this corollary characterizes the Brier-ERM specialization, while the reported operating point follows the stated F1-and-complexity objective. □

C FIXED-OBSERVATION VALIDATION OF THE INVERSE-STATE PROXY

The risk analysis above identifies a testable link between the inverse-state gap and calibration error. This section analyzes the prespecified fixed-observation Top200+DeFi mechanism setting whose S-ISP endpoint is .933/.0876. It does not re-estimate the final corpus-level results in Table 1. For each fixed prediction, we form

$$p_w = \sum_i w_i s_i, \qquad \Delta = C_1 - C_0, \qquad r = Y - p_w,$$

and estimate

$$A = \mathbb{E}[r\Delta], \qquad B = \mathbb{E}[\Delta^2], \qquad \mathrm{corr}(r, \Delta).$$

No additional model inference is used, and neither the fixed predictions nor the reported F1 and Brier scores are changed. Confidence intervals are obtained from 5,000 stratified bootstrap replicates that resample contracts within each vulnerability family. Because $\mathbb{E}[(p_w - Y)\Delta] = -A$, the negative values of A in Table 11 are equivalently positive alignment between the first-order calibration error $p_w - Y$ and its inverse-state proxy.

Table 11: Direct validation of the inverse-state proxy on fixed held-out predictions. Intervals are 95% stratified bootstrap intervals over contracts; the final column is the fraction of bootstrap replicates with $A < 0$.

Vulnerability	$\mathrm{corr}(Y - p_w, \Delta)$ [95% CI]	$A = \mathbb{E}[(Y - p_w)\Delta]$ [95% CI]	$\mathbb{E}[(p_w - Y)\Delta]$	Bootstrap $A < 0$
Logic error	-0.43 $[-0.60, -0.24]$	-0.030 $[-0.048, -0.012]$	$+0.030$	0.986
Access control	-0.51 $[-0.66, -0.35]$	-0.0345 $[-0.052, -0.017]$	$+0.0345$	0.998
Reentrancy	-0.04 $[-0.22, 0.14]$	-0.0015 $[-0.006, 0.003]$	$+0.0015$	0.612
Macro	-0.33 $[-0.47, -0.19]$	-0.0220 $[-0.036, -0.008]$	$+0.0220$	0.995

Logic error and access control have confidence intervals that exclude zero in the direction required by Proposition 1. The reentrancy estimate is statistically indistinguishable from zero; its development-selected choice $\gamma = 0$ therefore applies no test-time correction. The corresponding category F1 values remain .909, 1.000, and .889, and the Brier values remain .106, .050, and .107, yielding macro F1/Brier .933/.0876. This pattern links the theoretical sign condition to the selection behavior: the two reliably aligned families receive active corrections, whereas the family without reliable alignment remains at the decision-preserving boundary.

C.1 STEP-SIZE CONDITION AND SELECTED MECHANISM STRENGTHS

For the unprojected family, Theorem 1 gives

$$R_B(p_\gamma) - R_B(p_w) = \gamma A + \frac{\gamma^2}{4} B.$$

When $A < 0$ and $B > 0$, this quadratic decreases risk exactly for

$$0 < \gamma < -\frac{4A}{B}, \qquad \gamma^* = -\frac{2A}{B}.$$

Table 12 compares these quantities with the strengths chosen by development-only selection for this mechanism analysis. Both active strengths lie inside their category-specific decreasing intervals. For reentrancy, the interval shown is induced by a near-zero point estimate whose confidence interval includes zero; the selector conservatively deploys $\gamma = 0$.

Table 12: Unprojected theoretical step-size condition versus the selected S-ISP parameters on fixed predictions in the matched mechanism analysis. Observed changes use the projected and clipped predictor; theoretical changes use $\gamma A + \gamma^2 B/4$. For the Macro row, the theoretical change is the mean of the three category-specific changes under their selected strengths.

Vulnerability	A	B	γ^*	Decreasing interval	Selected γ	First-order Brier	S-ISP Brier	Observed ΔBrier / theoretical
Logic error	-0.030	0.040	1.50	$(0, 3.00)$	0.50	0.118	0.106	-0.012 / -0.0125
Access control	-0.0345	0.050	1.38	$(0, 2.76)$	1.00	0.072	0.050	-0.022 / -0.0220
Reentrancy	-0.0015	0.035	0.086	$(0, 0.171)$	0.00	0.107	0.107	0.000 / 0.000
Macro	-0.0220	0.0417	1.06	$(0, 2.11)$	0.5/1.0/0	0.099	0.0876	-0.0114 / -0.0115

The unprojected theoretical changes closely track the fixed-observation Brier differences for the active corrections. This agreement is empirical rather than an algebraic identity because the implemented predictor additionally applies the decision-preserving projection and clipping in Eqs. 5–6. The macro diagnostic $\gamma^* = 1.06$ and interval $(0, 2.11)$ describe the common-strength quadratic formed from macro-averaged A and B; they are not used to compute the macro theoretical change reported in the final column.

C.2 Exact projected-estimator risk decomposition

The preceding diagnostic concerns the unprojected family. We additionally evaluate the exact identity for the implemented estimator in this matched mechanism setting, including its projection and clipping. For every fixed held-out prediction, define

$$d = p_w - p_{\mathrm{SISP}}.$$

Theorem 2 gives

$$R_B(p_{\mathrm{SISP}}) - R_B(p_w) = 2\mathbb{E}[(Y - p_w)d] + \mathbb{E}[d^2]. \tag{18}$$

Table 13 reports every term by vulnerability family. The same 5,000 stratified bootstrap replicates resample contracts within each family. Because p_{SISP} in d already contains the implemented projection and clipping, Eq. 18 applies to the implemented predictions rather than an unprojected surrogate.

Table 13: Exact projected-estimator risk decomposition on fixed held-out predictions. Intervals are 95% stratified bootstrap intervals over contracts. Projection and clipping are included in $d = p_w - p_{\mathrm{SISP}}$.

Vulnerability	$2\mathbb{E}[(Y - p_w)d]$ [95% CI]	$\mathbb{E}[d^2]$ [95% CI]	$R_B(p_{\mathrm{SISP}}) - R_B(p_w)$ [95% CI]	$R_B(p_w)$	$R_B(p_{\mathrm{SISP}})$
Logic error	-0.0140 $[-0.021, -0.007]$	0.0020 $[0.0010, 0.0032]$	-0.0120 $[-0.018, -0.006]$	0.118	0.106
Access control	-0.0250 $[-0.033, -0.018]$	0.0030 $[0.0016, 0.0046]$	-0.0220 $[-0.030, -0.014]$	0.072	0.050
Reentrancy	0.0000 $[0, 0]$	0.0000 $[0, 0]$	0.0000 $[0, 0]$	0.107	0.107
Macro	-0.0130 $[-0.019, -0.007]$	0.0016 $[0.0008, 0.0025]$	-0.0114 $[-0.017, -0.005]$	0.099	0.0876

For logic error and access control, the negative residual-alignment term strictly dominates the squared correction, and the confidence intervals for the resulting risk differences exclude zero. Reentrancy has $d = 0$ because development-only selection chooses $\gamma = 0$, so every term in the identity is exactly zero. The macro Brier score decreases from 0.099 to 0.0876, while the decision-preserving projection leaves all category F1 values unchanged. These results close the empirical link between the risk identity and the implemented estimator rather than inferring its behavior from the unprojected quadratic alone.

C.3 Connecting the implemented gap to calibration bias

Proposition 4 supplies sufficient conditions under which the implemented $\Delta = C_1 - C_0$ is a positive proxy for the first-order calibration bias $b = p_w - \eta$. In particular,

$$\mathbb{E}[(Y - p_w)\Delta] = -\mathbb{E}[b\Delta] < 0.$$

The fixed-observation estimates in Table 11 provide direct empirical support: $\mathbb{E}[(p_w - Y)\Delta]$ is $+0.030$, $+0.0345$, $+0.0015$, and $+0.0220$ for logic error, access control, reentrancy, and the macro statistic, respectively. The confidence intervals exclude zero for logic error, access control, and the macro statistic. Reentrancy is statistically indistinguishable from zero, consistently with the development-only selector choosing $\gamma = 0$ for that family and positive strengths for the two reliably aligned families.

D Compute accounting and case studies

D.1 Rule-search trajectory

Figure 2 reports the measured category-level transition from the initial rule-bank configuration to the final S-ISP endpoint reported in Table 1. Logic-error and reentrancy detection improve, while

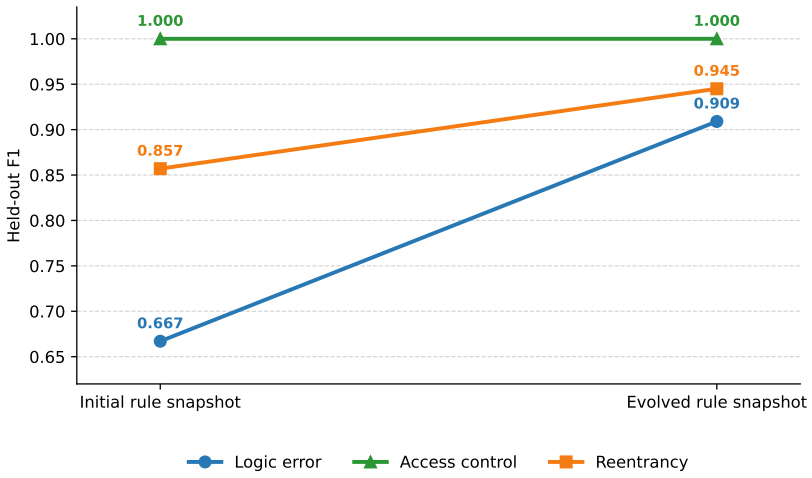

Figure 2: Measured held-out F1 at the initial rule-bank configuration and the final S-ISP endpoint reported in Table 1; connecting segments link the two recorded checkpoints.

access-control performance is preserved. The figure connects the two recorded endpoints for comparison.

We report aggregate GPU-hours, the sum of device wall times divided by 3600; this accounting is invariant to whether jobs run sequentially or concurrently on seven cards. Its scope is the directly comparable inference workload: baseline calls, four-auditor cache construction, and the selected sparse deployment budget. Offline rule construction is a separate development-stage activity and is not mixed into these per-contract inference values. On the common confirmation workload, measured GPU-hours are 0.113 (D-Llama), 0.158 (DME), 0.191 (GPTLens), 0.085 (LMM), 0.315 (Similar), 0.194 (Audit), and 0.573 (SmartLLM). Their calls per contract are 1.00, 1.00, 1.06, 1.00, 3.00, 1.63, and 3.00. Four-auditor S-ISP cache construction costs 0.569 GPU-hours; the selected two-auditor deployment budget is approximately half (0.284 GPU-hours), and aggregation itself costs zero GPU-hours and zero model calls. These are empirical A800-80GB measurements, not FLOP estimates.

D.2 Full contracts for the reentrancy case study

The following two contracts are reproduced in full, rather than reduced to isolated statements. The first is a capability-exposure false negative for a canonical concept prompt; the second is an authorization-protected false positive for a surface-pattern detector. Keeping inheritance, modifiers, state, and every public entry point is essential because the relevant witness and refuter are cross-function properties.

D.2.1 Vulnerable but Easily Missed Example

```
pragma solidity ^0.4.24;

library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
    // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
    if (a == 0) {
      return 0;
    }

    c = a * b;
    assert(c / a == b);
```

```
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    // uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return a / b;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend
    ).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
    c = a + b;
    assert(c >= a);
    return c;
  }
}

contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

  uint256 totalSupply_;

  /**
  * @dev total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address whose balance is queried.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public view returns (uint256) {
    return balances[_owner];
  }

}

contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender)
```

```
    public view returns (uint256);

  function transferFrom(address from, address to, uint256 value)
    public returns (bool);

  function approve(address spender, uint256 value) public returns (bool);
  event Approval(
    address indexed owner,
    address indexed spender,
    uint256 value
  );
}

contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;


  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  function transferFrom(
    address _from,
    address _to,
    uint256 _value
  )
    public
    returns (bool)
  {
    require(_to != address(0));
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfer(_from, _to, _value);
    return true;
  }

  /**
   * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.
     sender.
   *
   * Beware that changing an allowance with this method brings the risk that someone may use
     both the old
   * and the new allowance by unfortunate transaction ordering. One possible solution to
     mitigate this
   * race condition is to first reduce the spender's allowance to 0 and set the desired value
     afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   * @param _spender The address which will spend the funds.
   * @param _value The amount of tokens to be spent.
   */
  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

  /**
   * @dev Function to check the amount of tokens that an owner allowed to a spender.
   * @param _owner address The address which owns the funds.
   * @param _spender address The address which will spend the funds.
   * @return A uint256 specifying the amount of tokens still available for the spender.
   */
  function allowance(
    address _owner,
    address _spender
  )
    public
    view
    returns (uint256)
  {
    return allowed[_owner][_spender];
  }

  /**
```

```
   * @dev Increase the amount of tokens that an owner allowed to a spender.
   *
   * approve should be called when allowed[_spender] == 0. To increment
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _addedValue The amount of tokens to increase the allowance by.
   */
  function increaseApproval(
    address _spender,
    uint _addedValue
  )
    public
    returns (bool)
  {
    allowed[msg.sender][_spender] = (
      allowed[msg.sender][_spender].add(_addedValue));
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

  /**
   * @dev Decrease the amount of tokens that an owner allowed to a spender.
   *
   * approve should be called when allowed[_spender] == 0. To decrement
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _subtractedValue The amount of tokens to decrease the allowance by.
   */
  function decreaseApproval(
    address _spender,
    uint _subtractedValue
  )
    public
    returns (bool)
  {
    uint oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue > oldValue) {
      allowed[msg.sender][_spender] = 0;
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}

contract ERC827 is ERC20 {
  function approveAndCall(
    address _spender,
    uint256 _value,
    bytes _data
  )
    public
    payable
    returns (bool);

  function transferAndCall(
    address _to,
    uint256 _value,
    bytes _data
  )
    public
    payable
    returns (bool);

  function transferFromAndCall(
    address _from,
    address _to,
    uint256 _value,
    bytes _data
  )
    public
    payable
    returns (bool);
}
```

```
contract ERC827Token is ERC827, StandardToken {

  /**
   * @dev Addition to ERC20 token methods. It allows to
   * @dev approve the transfer of value and execute a call with the sent data.
   *
   * @dev Beware that changing an allowance with this method brings the risk that
   * @dev someone may use both the old and the new allowance by unfortunate
   * @dev transaction ordering. One possible solution to mitigate this race condition
   * @dev is to first reduce the spender's allowance to 0 and set the desired value
   * @dev afterwards:
   * @dev https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   *
   * @param _spender The address that will spend the funds.
   * @param _value The amount of tokens to be spent.
   * @param _data ABI-encoded contract call to call `_to` address.
   *
   * @return true if the call function was executed successfully
   */
  function approveAndCall(
    address _spender,
    uint256 _value,
    bytes _data
  )
    public
    payable
    returns (bool)
  {
    require(_spender != address(this));

    super.approve(_spender, _value);

    // solium-disable-next-line security/no-call-value
    require(_spender.call.value(msg.value)(_data));

    return true;
  }

  /**
   * @dev Addition to ERC20 token methods. Transfer tokens to a specified
   * @dev address and execute a call with the sent data on the same transaction
   *
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amout of tokens to be transfered
   * @param _data ABI-encoded contract call to call `_to` address.
   *
   * @return true if the call function was executed successfully
   */
  function transferAndCall(
    address _to,
    uint256 _value,
    bytes _data
  )
    public
    payable
    returns (bool)
  {
    require(_to != address(this));

    super.transfer(_to, _value);

    // solium-disable-next-line security/no-call-value
    require(_to.call.value(msg.value)(_data));
    return true;
  }

  /**
   * @dev Addition to ERC20 token methods. Transfer tokens from one address to
   * @dev another and make a contract call on the same transaction
   *
   * @param _from The address which you want to send tokens from
   * @param _to The address which you want to transfer to
   * @param _value The amout of tokens to be transferred
   * @param _data ABI-encoded contract call to call `_to` address.
   *
   * @return true if the call function was executed successfully
   */
  function transferFromAndCall(
    address _from,
    address _to,
    uint256 _value,
```

```
    bytes _data
  )
    public payable returns (bool)
  {
    require(_to != address(this));

    super.transferFrom(_from, _to, _value);

    // solium-disable-next-line security/no-call-value
    require(_to.call.value(msg.value)(_data));
    return true;
  }

  /**
   * @dev Addition to StandardToken methods. Increase the amount of tokens that
   * @dev an owner allowed to a spender and execute a call with the sent data.
   *
   * @dev approve should be called when allowed[_spender] == 0. To increment
   * @dev allowed value is better to use this function to avoid 2 calls (and wait until
   * @dev the first transaction is mined)
   * @dev From MonolithDAO Token.sol
   *
   * @param _spender The address which will spend the funds.
   * @param _addedValue The amount of tokens to increase the allowance by.
   * @param _data ABI-encoded contract call to call `_spender` address.
   */
  function increaseApprovalAndCall(
    address _spender,
    uint _addedValue,
    bytes _data
  )
    public
    payable
    returns (bool)
  {
    require(_spender != address(this));

    super.increaseApproval(_spender, _addedValue);

    // solium-disable-next-line security/no-call-value
    require(_spender.call.value(msg.value)(_data));

    return true;
  }

  /**
   * @dev Addition to StandardToken methods. Decrease the amount of tokens that
   * @dev an owner allowed to a spender and execute a call with the sent data.
   *
   * @dev approve should be called when allowed[_spender] == 0. To decrement
   * @dev allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * @dev From MonolithDAO Token.sol
   *
   * @param _spender The address which will spend the funds.
   * @param _subtractedValue The amount of tokens to decrease the allowance by.
   * @param _data ABI-encoded contract call to call `_spender` address.
   */
  function decreaseApprovalAndCall(
    address _spender,
    uint _subtractedValue,
    bytes _data
  )
    public
    payable
    returns (bool)
  {
    require(_spender != address(this));

    super.decreaseApproval(_spender, _subtractedValue);

    // solium-disable-next-line security/no-call-value
    require(_spender.call.value(msg.value)(_data));

    return true;
  }

}

contract AtripToken is ERC827Token {
   using SafeMath for uint256;
```

```
    string public name = "Atrip Token";
    string public symbol = "APK";
    uint public decimals = 18;

    address public wallet = 0x0;//

    constructor (address _wallet) public  {
        wallet = _wallet;
        totalSupply_ = 21 * 100000000 * 10 ** decimals;
        balances[wallet] = totalSupply_;
     }

     /**
     * Do not allow direct deposits.
     */
     function() public{
         revert();
     }
}
```

D.2.2 NON-VULNERABLE BUT EASILY OVER-REPORTED EXAMPLE

```
pragma solidity ^0.4.24;
pragma experimental "v0.5.0";
pragma experimental ABIEncoderV2;

library Math {

  struct Fraction {
    uint256 numerator;
    uint256 denominator;
  }

  function isPositive(Fraction memory fraction) internal pure returns (bool) {
    return fraction.numerator > 0 && fraction.denominator > 0;
  }

  function mul(uint256 a, uint256 b) internal pure returns (uint256 r) {
    r = a * b;
    require((a == 0) || (r / a == b));
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256 r) {
    r = a / b;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256 r) {
    require((r = a - b) <= a);
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256 r) {
    require((r = a + b) >= a);
  }

  function min(uint256 x, uint256 y) internal pure returns (uint256 r) {
    return x <= y ? x : y;
  }

  function max(uint256 x, uint256 y) internal pure returns (uint256 r) {
    return x >= y ? x : y;
  }

  function mulDiv(uint256 value, uint256 m, uint256 d) internal pure returns (uint256 r) {
    r = value * m;
    if (r / value == m) {
      r /= d;
    } else {
      r = mul(value / d, m);
    }
  }

  function mulDivCeil(uint256 value, uint256 m, uint256 d) internal pure returns (uint256 r) {
    r = value * m;
    if (r / value == m) {
      if (r % d == 0) {
        r /= d;
      } else {
        r = (r / d) + 1;
```

```
      }
    } else {
      r = mul(value / d, m);
      if (value % d != 0) {
        r += 1;
      }
    }
  }

  function mul(uint256 x, Fraction memory f) internal pure returns (uint256) {
    return mulDiv(x, f.numerator, f.denominator);
  }

  function mulCeil(uint256 x, Fraction memory f) internal pure returns (uint256) {
    return mulDivCeil(x, f.numerator, f.denominator);
  }

  function div(uint256 x, Fraction memory f) internal pure returns (uint256) {
    return mulDiv(x, f.denominator, f.numerator);
  }

  function divCeil(uint256 x, Fraction memory f) internal pure returns (uint256) {
    return mulDivCeil(x, f.denominator, f.numerator);
  }

  function mul(Fraction memory x, Fraction memory y) internal pure returns (Math.Fraction) {
    return Math.Fraction({
      numerator: mul(x.numerator, y.numerator),
      denominator: mul(x.denominator, y.denominator)
    });
  }
}

contract FsTKColdWallet {
  using Math for uint256;

  event ConfirmationNeeded(address indexed initiator, bytes32 indexed operation, address
     indexed to, uint256 value, bytes data);
  event Confirmation(address indexed authority, bytes32 indexed operation);
  event Revoke(address indexed authority, bytes32 indexed operation);

  event AuthorityChanged(address indexed oldAuthority, address indexed newAuthority);
  event AuthorityAdded(address authority);
  event AuthorityRemoved(address authority);

  event RequirementChanged(uint256 required);
  event DayLimitChanged(uint256 dayLimit);
  event SpentTodayReset(uint256 spentToday);

  event Deposit(address indexed from, uint256 value);
  event SingleTransaction(address indexed authority, address indexed to, uint256 value, bytes
     data, address created);
  event MultiTransaction(address indexed authority, bytes32 indexed operation, address indexed
      to, uint256 value, bytes data, address created);

  struct TransactionInfo {
    address to;
    uint256 value;
    bytes data;
  }

  struct PendingTransactionState {
    TransactionInfo info;
    uint256 confirmNeeded;
    uint256 confirmBitmap;
    uint256 index;
  }

  modifier onlyAuthority {
    require(isAuthority(msg.sender));
    _;
  }

  modifier confirmAndRun(bytes32 operation) {
    if (confirmAndCheck(operation)) {
      _;
    }
  }

  uint256 constant MAX_AUTHORITIES = 250;
```

```
uint256 public requiredAuthorities;
uint256 public numAuthorities;

uint256 public dailyLimit;
uint256 public spentToday;
uint256 public lastDay;

address[256] public authorities;
mapping(address => uint256) public authorityIndex;
mapping(bytes32 => PendingTransactionState) public pendingTransaction;
bytes32[] public pendingOperation;

constructor(address[] _authorities, uint256 required, uint256 _daylimit) public {
  require(
    required > 0 &&
    authorities.length >= required
  );

  numAuthorities = _authorities.length;
  for (uint256 i = 0; i < _authorities.length; i += 1) {
    authorities[1 + i] = _authorities[i];
    authorityIndex[_authorities[i]] = 1 + i;
  }

  requiredAuthorities = required;

  dailyLimit = _daylimit;
  lastDay = today();
}

function() external payable {
  if (msg.value > 0) {
    emit Deposit(msg.sender, msg.value);
  }
}

function getAuthority(uint256 index) public view returns (address) {
  return authorities[index + 1];
}

function getAuthorityIndex(address authority) public view returns (uint256 index) {
  index = authorityIndex[authority];
  require(index > 0);
}

function isAuthority(address authority) public view returns (bool) {
  return authorityIndex[authority] > 0;
}

function hasConfirmed(bytes32 operation, address _address) public view returns (bool) {
  return (pendingTransaction[operation].confirmBitmap & (1 << getAuthorityIndex(_address)))
   != 0;
}

function changeAuthority(address from, address to) public confirmAndRun(keccak256(msg.data))
    {
  require(!isAuthority(to));

  uint256 index = getAuthorityIndex(from);
  authorities[index] = to;
  authorityIndex[to] = index;
  delete authorityIndex[from];
  clearPending();

  emit AuthorityChanged(from, to);
}

function addAuthority(address authority) public confirmAndRun(keccak256(msg.data)) {
  require(!isAuthority(authority));
  if (numAuthorities >= MAX_AUTHORITIES) {
    reOrganizeAuthorities();
  }
  require(numAuthorities < MAX_AUTHORITIES);

  numAuthorities += 1;
  authorities[numAuthorities] = authority;
  authorityIndex[authority] = numAuthorities;
  clearPending();

  emit AuthorityAdded(authority);
}
```

```
function removeAuthority(address authority) public confirmAndRun(keccak256(msg.data)) {
  require(numAuthorities > requiredAuthorities);

  uint256 index = getAuthorityIndex(authority);
  delete authorities[index];
  delete authorityIndex[authority];
  clearPending();
  reOrganizeAuthorities();

  emit AuthorityRemoved(authority);
}

function setRequirement(uint256 required) public confirmAndRun(keccak256(msg.data)) {
  require(numAuthorities >= requiredAuthorities);
  clearPending();

  emit RequirementChanged(requiredAuthorities = required);
}

function setDailyLimit(uint256 _dailyLimit) public confirmAndRun(keccak256(msg.data)) {
  clearPending();

  emit DayLimitChanged(dailyLimit = _dailyLimit);
}

function resetSpentToday() public confirmAndRun(keccak256(msg.data)) {
  clearPending();

  emit SpentTodayReset(spentToday);
  delete spentToday;
}

function propose(
  address to,
  uint256 value,
  bytes data
)
  public
  onlyAuthority
  returns (bytes32 operation)
{
  if ((data.length == 0 && checkAndUpdateLimit(value)) || requiredAuthorities == 1) {
    emit SingleTransaction(msg.sender, to, value, data, execute0(to, value, data));
  } else {
    operation = keccak256(abi.encodePacked(msg.data, pendingOperation.length));
    PendingTransactionState storage status = pendingTransaction[operation];
    if (status.info.to == 0 && status.info.value == 0 && status.info.data.length == 0) {
      status.info = TransactionInfo({
        to: to,
        value: value,
        data: data
      });
    }

    if (!confirm(operation)) {
      emit ConfirmationNeeded(msg.sender, operation, to, value, data);
    }
  }
}

function revoke(bytes32 operation) public {
  uint256 confirmFlag = 1 << getAuthorityIndex(msg.sender);
  PendingTransactionState storage state = pendingTransaction[operation];
  if (state.confirmBitmap & confirmFlag > 0) {
    state.confirmNeeded += 1;
    state.confirmBitmap &= ~confirmFlag;
    emit Revoke(msg.sender, operation);
  }
}

function confirm(bytes32 operation) public confirmAndRun(operation) returns (bool) {
   PendingTransactionState storage status = pendingTransaction[operation];
  if (status.info.to != 0 || status.info.value != 0 || status.info.data.length != 0) {
    emit MultiTransaction(
      msg.sender,
      operation,
      status.info.to,
      status.info.value,
      status.info.data,
      execute0(status.info.to, status.info.value, status.info.data)
```

```
    );
    delete pendingTransaction[operation].info;

    return true;
  }
}

function execute0(
  address to,
  uint256 value,
  bytes data
)
  private
  returns (address created)
{
  if (to == 0) {
    created = create0(value, data);
  } else {
    require(to.call.value(value)(data));
  }
}

function create0(uint256 value, bytes code) internal returns (address _address) {
  assembly {
    _address := create(value, add(code, 0x20), mload(code))
    if iszero(extcodesize(_address)) {
      revert(0, 0)
    }
  }
}

function confirmAndCheck(bytes32 operation) private returns (bool) {
  PendingTransactionState storage pending = pendingTransaction[operation];
  if (pending.confirmNeeded == 0) {
    pending.confirmNeeded = requiredAuthorities;
    delete pending.confirmBitmap;
    pending.index = pendingOperation.length;
    pendingOperation.push(operation);
  }

  uint256 confirmFlag = 1 << getAuthorityIndex(msg.sender);

  if (pending.confirmBitmap & confirmFlag == 0) {
    emit Confirmation(msg.sender, operation);
    if (pending.confirmNeeded <= 1) {
      delete pendingOperation[pending.index];
      delete pending.confirmNeeded;
      delete pending.confirmBitmap;
      delete pending.index;
      return true;
    } else {
      pending.confirmNeeded -= 1;
      pending.confirmBitmap |= confirmFlag;
    }
  }
}

function checkAndUpdateLimit(uint256 value) private returns (bool) {
  if (today() > lastDay) {
    spentToday = 0;
    lastDay = today();
  }

  uint256 _spentToday = spentToday.add(value);
  if (_spentToday <= dailyLimit) {
    spentToday = _spentToday;
    return true;
  }
  return false;
}

function today() private view returns (uint256) {
  return block.timestamp / 1 days;
}

function reOrganizeAuthorities() private {
  uint256 free = 1;
  while (free < numAuthorities) {
    while (free < numAuthorities && authorities[free] != 0) {
      free += 1;
    }
```

```
      while (numAuthorities > 1 && authorities[numAuthorities] == 0) {
        numAuthorities -= 1;
      }
      if (free < numAuthorities && authorities[numAuthorities] != 0 && authorities[free] == 0)
      {
        authorities[free] = authorities[numAuthorities];
        authorityIndex[authorities[free]] = free;
        delete authorities[numAuthorities];
      }
    }
  }

  function clearPending() private {
    for (uint256 i = 0; i < pendingOperation.length; i += 1) {
      delete pendingTransaction[pendingOperation[i]];
    }
    delete pendingOperation;
  }

}
```

D.3 Case Analysis

Vulnerable case: callback after capability exposure. The first contract is vulnerable but does not match the most common reentrancy template. The core ERC20 transfer logic in `BasicToken` and `StandardToken` updates balances and allowances before returning. There is no direct `withdraw` function that sends Ether before clearing a balance. A detector guided only by the canonical phrase "external call before state update" may therefore assign a low risk score.

The key risk appears in the ERC827-style extension functions. For example, `approveAndCall` first invokes `super.approve(_spender, _value)`, which writes `allowed[msg.sender][_spender] = _value`. It then performs an arbitrary low-level external call through `_spender.call.value(msg.value)(_data)`. Since `_spender` is externally supplied and may be a malicious contract, the callee can execute arbitrary callback logic before `approveAndCall` returns. During this callback, the callee may reenter public token methods such as `transferFrom`, observe the newly granted allowance, and consume it immediately. Similar callback surfaces appear in `increaseApprovalAndCall`, `decreaseApprovalAndCall`, `transferAndCall`, and `transferFromAndCall`, where token or allowance state transitions are followed by a user-controlled external call.

This case is difficult because the unsafe pattern is not simply "effect after interaction". Instead, the effect itself exposes a capability that becomes exploitable during the subsequent interaction. The vulnerability therefore depends on cross-function reentry, inherited ERC20 logic, and authorization-like token allowance semantics. Such samples are also relatively sparse and discrete: they may not share strong lexical similarity with canonical Ether-withdrawal examples, and nearest-neighbor retrieval may fail to retrieve useful analogues. Baseline LLM detectors that rely mainly on the normative concept document or retrieved similar contracts can therefore miss the vulnerability. In our framework, this kind of false negative is converted into an evolved rule: after allowance changes or token-state changes, hook-style external callbacks must be analyzed as reentrancy-relevant because the callee may reenter `transferFrom`, `approve`, or other public token methods using the newly exposed capability.

Non-vulnerable case: low-level call protected by authorization and pre-call state handling. The second contract is non-vulnerable under the target reentrancy semantics, but it contains several syntactic features that resemble vulnerable code. In particular, `execute0` performs a low-level value transfer through `to.call.value(value)(data)`, and some cleanup appears after this call. A detector that over-relies on the presence of `call.value` or post-call deletion may therefore produce a false positive.

A closer control-flow and authorization analysis shows why this sample should not be labeled as a reentrancy vulnerability. The private function `execute0` is reachable through `propose` or `confirm`. In the single-transaction branch of `propose`, the caller must satisfy `onlyAuthority`. If the daily-limit path is used, `checkAndUpdateLimit(value)` updates `spentToday` before the external call, so a reentrant execution cannot reuse stale daily-limit state to bypass the spending constraint. In the multi-signature path, execution is guarded by

confirmAndRun, whose internal routine confirmAndCheck requires the caller to be an authority and clears the confirmation counter, bitmap, and pending-operation index before the external execution. Thus, an arbitrary external callee cannot reenter and execute the same operation again, because it cannot pass the same authority and confirmation requirements. If the callee is itself an authorized authority, the issue becomes part of the wallet's intended authorization model rather than an unauthorized reentrancy drain caused by stale state.

This example exposes the failure mode opposite to that in the ERC827 case. The contract appears suspicious because it contains a low-level external call, Ether transfer, callback-capable target, and delayed cleanup. However, authority checks, threshold confirmation, and pre-call spending-limit updates already protect the relevant security state. A concept-level document does not explicitly teach the detector to distinguish security-critical invalidation from non-critical bookkeeping cleanup or to verify whether the reentrant callee can pass the same authorization gate. Retrieval is also unreliable because superficially similar call.value patterns may be vulnerable or safe depending on their surrounding control logic. Our method addresses this issue by evolving negative rules from false positives: rather than reporting reentrancy from the low-level call pattern alone, the detector must first establish a feasible reentrant path that can reuse uncleared security-critical state and pass the necessary guards.

D.4 Example Evolved Rule Group

The following rule group presents the operational knowledge produced by error-driven rule evolution. It is more specific than the original concept document while remaining compact enough to serve as rule-conditioned expert context.

1. Check whether the contract performs a low-level call, Ether transfer, token callback, or any other external interaction that may transfer control to an untrusted contract.
2. Before every external call, identify all security-critical state variables that may affect balances, allowances, execution status, ownership, spending limits, or pending-operation states.
3. A contract is reentrancy-risky if an external call is made before the relevant state has been updated, invalidated, locked, or otherwise made safe against repeated execution.
4. Do not limit the analysis to local balance updates. Also inspect whether the pre-call state grants a capability, such as an allowance, withdrawal right, execution permission, or callback-enabled token operation, that can be exploited during reentry.
5. If only part of the relevant state is locked before the external call while other related state remains reusable, treat this as a potential reentrancy boundary case.
6. Check whether the contract uses an explicit reentrancy guard, execution flag, processed marker, or equivalent lock, and verify that the lock is set before the external call and released only after the unsafe region.
7. If the external call is read-only, cannot invoke user-controlled code, or targets a verified and trusted contract under the threat model, reduce the reentrancy risk accordingly.
8. If the callee is user-controlled, passed as a function argument, derived from msg.sender, or can contain arbitrary fallback logic, treat the call target as potentially malicious unless the code proves otherwise.
9. When a function updates state before an external call, still check whether the update itself enables a harmful callback, such as granting allowance before calling the approved spender.
10. For ERC827-style approveAndCall, transferAndCall, transferFromAndCall, or similar hook-based token functions, treat an external callback after allowance or token-state changes as reentrancy-relevant, because the callee may reenter transferFrom, approve, or other public token methods using the newly exposed capability.
11. If cleanup occurs after an external call, distinguish security-critical invalidation from non-critical bookkeeping. Delayed cleanup is dangerous only when the reentrant caller can reuse the uncleared state to repeat a protected action.

12. Do not classify a contract as vulnerable merely because it contains `call.value`, `call`, `send`, or `transfer`. First verify that a feasible reentrant path can reenter a sensitive public function and pass its guards.
13. For multi-signature wallets or authority-gated execution contracts, check whether the reentrant callee can satisfy the same authority or confirmation requirements. If the reentrant path is blocked by authorization, threshold confirmation, or pre-call state invalidation, avoid reporting reentrancy solely from the low-level call pattern.
14. Inspect cross-function reentry, not only same-function reentry. A vulnerability may arise when the external callback enters a different public function that reads or consumes stale state.
15. Inspect inherited contracts and parent methods. A child function may appear safe locally but may expose reentrancy through a state update in `super` followed by an external call in the child method.
16. Check whether state updates around the external call are atomic with respect to the vulnerable action. If the action can be repeated before all relevant state transitions complete, assign a higher risk score.
17. Calls made in constructors or purely read-only functions usually do not create reentrancy risk for persistent contract state, unless the constructor publishes control to an already deployed external contract or the function is incorrectly marked and can modify state through external effects.
18. When evidence is mixed, assign a continuous risk score rather than making an immediate binary decision. Increase the score for feasible attacker-controlled callbacks and reusable state, and decrease it for trusted callees, complete pre-call invalidation, effective locks, or infeasible reentrant authorization paths.

These rules demonstrate the role of rule evolution in S-ISP. The false-negative ERC827 case contributes a positive rule about callbacks after capability exposure, whereas the false-positive cold-wallet case contributes a negative rule about authorization-gated low-level calls and non-critical post-call cleanup. Such rules are difficult to derive from the initial concept document alone and equally difficult to recover reliably through lexical retrieval because the relevant distinction is semantic rather than purely textual.

D.5 Prompt Design for Rule Evolution and Expert Auditing

Rule evolution prompt. The following prompt is used to refine vulnerability-specific rules from a misclassified case. The template field {`definition.definition`} contains the current high-level definition; {`definition.technical_description`} contains the current technical explanation and examples; {`definition.detection_criteria`} contains the current operational rule set; {`error_type`} indicates false positive or false negative; {`error_case['fname']`} is the file name; {`error_case['true_label']`} is the ground-truth label; {`error_case['pred_score']`} is the model-predicted risk score; {`error_case['evidence']`} is the evidence generated by the previous detector; {`error_case['code_snippet']`} is the Solidity fragment; and {`importance_text`} optionally emphasizes the importance or frequency of the error case.

```
You are a smart contract security expert. You are optimizing the detection rules for
    reentrancy vulnerabilities. The currently used vulnerability definition is as follows:

Definition: {definition.definition}

Technical description: {definition.technical_description}

Detection criteria: {definition.detection_criteria}

Below is a case where the model made an incorrect prediction:

Misclassified case ({error_type})

File name: {error_case['fname']}

Ground-truth label: {error_case['true_label']} ({'vulnerable' if error_case['true_label']==1
    else 'non-vulnerable'})
```

```
Model-predicted score: {error_case['pred_score']:.2f}

Model evidence: {error_case['evidence']}

Contract code snippet:
```solidity
{error_case['code_snippet']}
```

{importance_text}

Based on this misclassified case, analyze the limitations of the current detection rules and
     propose improved concrete detection rules. The improved rules should help correctly
     handle similar cases in the future.

Output requirements: Return the result in JSON format with the following fields:
- "definition": the vulnerability definition, which may remain unchanged or be slightly
     revised;
- "technical_description": the technical description, which may remain unchanged or be
     slightly revised;
- "detection_criteria": the improved concrete and actionable detection criteria. Each rule
     should follow a pattern similar to "Check whether the code contains XXX; if XXX holds,
     then it tends to indicate XXX", where XXX refers to a specific code pattern or semantic
     condition. Please list the criteria as multi-line text, with one checking point per line,
      and merge similar or related points into the same line.

Make sure the returned JSON is valid and wrapped in ```json.

Improved definition:
```

Rule merging prompt. After multiple rule suggestions are generated from different error cases, the following prompt merges them into a unified rule group.

```
You are a smart contract security expert. You are integrating improvement suggestions from
     multiple misclassified cases into a unified set of detection rules. The currently used
     vulnerability definition is as follows:

Definition: {original_definition.definition}

Technical description: {original_definition.technical_description}

Detection criteria: {original_definition.detection_criteria}

Below are the improvement suggestions obtained from different misclassified cases. Each
     suggestion may focus on a different aspect:

{suggestions_text}

Please synthesize the above suggestions into a unified set of detection rules. The merged
     rules should cover the improvement points from all cases, remain logically consistent,
     and be as clear and actionable as possible.

Output requirements: Return the result in JSON format with the following fields:
- "definition": the vulnerability definition, which may remain unchanged or be slightly
     revised;
- "technical_description": the technical description, which may remain unchanged or be
     slightly revised;
- "detection_criteria": the merged concrete detection rules. The granularity should be fine
     enough to support judgment based on code content. Each rule should follow a pattern
     similar to "Check whether the code contains XXX; if XXX holds, then it tends to indicate
     XXX", where XXX refers to a specific code pattern or semantic condition. Please list the
     rules as multi-line text, with one checking point per line. Merge similar or related
     points into the same line. The total number of points should not exceed thirty.

Make sure the returned JSON is valid and wrapped in ```json.

Merged definition:
```

Initial reentrancy context. The following initial context is instantiated in the rule-evolution prompt before any error-driven refinement.

```
Definition:
SC05:2025 -- Reentrancy. A reentrancy attack exploits a smart contract vulnerability where a
     function makes an external call to another contract before properly updating or
     protecting its own security-critical state. This allows the external contract, possibly
```

```
    malicious, to reenter the original contract and repeat certain actions, such as
    withdrawals or authorization-dependent operations, using stale or reusable state. Such
    attacks may allow an attacker to drain funds, trigger unauthorized execution, or
    repeatedly consume contract resources beyond the intended limit.

Technical description:
A reentrancy vulnerability occurs when a smart contract transfers control to an external
    contract while some relevant internal state remains reusable, inconsistent, or
    insufficiently protected. The attacker can use a malicious fallback function or callback
    to reenter the victim contract before the original execution finishes. In the canonical
    example, a withdraw function sends Ether to msg.sender before setting balances[msg.sender
    ] to zero, allowing the attacker to repeatedly withdraw using the same balance. The
    impact may include draining the contract balance, executing unauthorized actions, or
    violating accounting and authorization invariants.

Initial detection criteria:
Check whether all relevant state variables are updated, invalidated, or locked before external
     calls such as call, delegatecall, transfer, or send. Check whether the contract follows
    the Checks-Effects-Interactions pattern. Check whether a reentrancy lock, such as
    OpenZeppelin's ReentrancyGuard, is used correctly. Recommended fixes include updating all
     internal state before calling external contracts and using a well-tested reentrancy
    guard for functions that transfer control to untrusted contracts.
```

Expert auditing prompt. The following prompt is used by each rule-conditioned expert to produce a continuous risk score rather than an immediate binary judgment. The template variable {role_description} defines the expert role; {agent_context} contains the vulnerability name, definition, technical description, and evolved detection rules; and {code} is the Solidity contract under analysis.

```
{role_description}

## Vulnerability to be detected

{agent_context}

## Smart contract code to be analyzed

'''solidity
{code}
'''

## Analysis task

Please assess the risk only for the vulnerability described above. Do not evaluate other
    vulnerability types.

## Output format requirements

Please return the analysis result in JSON format and wrap the JSON content with '''json. The
    following is an example:

'''json
{
  "vulnerability_type": "reentrancy",
  "risk_score": 0.50,
  "evidence": [
    "External call is performed before state update",
    "No reentrancy guard is used"
  ],
  "summary": "The contract has a potential reentrancy risk."
}
'''

Please ensure that:
1. The returned format must be strict JSON and wrapped with '''json;
2. risk_score must be a floating-point number between 0.0 and 1.0, rounded to two decimal
    places;
3. evidence must be a list of strings containing concrete evidence, with at least one item;
4. summary must briefly assess the vulnerability risk.

When making the judgment, refer to the given detection rules. For the score distribution, a
    score above 0.5 indicates relatively significant vulnerability risk, while a score below
    0.5 indicates relatively low vulnerability risk.

During the analysis, carefully reason about whether the suspected vulnerability is actually
    feasible before assigning the score. Do not rigidly match isolated words in the criteria,
     and avoid false judgments caused by either overly strict or overly superficial
    interpretation.
```

```
Now begin the analysis:
```

Role description and reentrancy context for expert auditing.

```
Role description:
You are a professional smart contract security auditor. Please analyze the given smart
     contract code and determine whether it contains the vulnerability defined below.

Agent context for reentrancy:
Name: Reentrancy vulnerability.

Definition: SC05:2025 -- Reentrancy. A reentrancy attack exploits a smart contract
     vulnerability where a function makes an external call to another contract before properly
      updating or protecting its own security-critical state. This allows the external
     contract, possibly malicious, to reenter the original contract and repeat certain actions
     , such as withdrawals, using the same stale state. Through such attacks, an attacker may
     drain funds from the contract or trigger unauthorized actions.

Technical description: A reentrancy attack occurs when a smart contract function performs an
     external call before the relevant internal state has been safely updated, invalidated, or
      locked. The attacker can exploit this by using a malicious contract to call back into
     the original contract during the external call, causing sensitive operations such as
     withdrawals or authorization-dependent transfers to be executed repeatedly. The most
     direct impact is fund loss: attackers may withdraw more assets than they are entitled to
     and potentially empty the contract balance. Reentrancy may also trigger unintended or
     unauthorized function calls in the contract or related systems.

Reference detection rules: Check whether all relevant state variables are updated before
     external calls such as call, delegatecall, send, or transfer; check whether the contract
     follows the Checks-Effects-Interactions pattern; check whether an effective reentrancy
     guard is used; and analyze whether a feasible callback path can reenter a sensitive
     public function using stale, reusable, or newly exposed security-critical state.
```

D.6 End-to-end trace through evolution and S-ISP

The vulnerable contract above most clearly demonstrates the complete chain. The seed criterion searches for an external call followed by a protective state write and therefore misses the case: the allowance write occurs first. Error attribution identifies the missing premise as *capability exposure* rather than statement order. Reflection proposes a positive rule that enumerates attacker-controlled callbacks after authority-expanding writes and follows every inherited public entry point that can consume the new authority. The paired benign wallet contributes the refuter that such a callback is harmless when all replay-enabling authorization facts have already been invalidated. The candidate pair is retained only after the next audit–compare pass repairs both recorded errors without a development regression. The resulting rule bank therefore encodes a semantic boundary rather than the names of these two contracts.

Figure 1 includes a worked calculation that exposes every deterministic aggregation step. The displayed four-auditor score vector is $s = (0.85, 0.30, 0.75, 0.20)$, and its development-selected support is $G = \{1, 3\}$ and $w_1 = w_3 = 1/2$. The first-order probability is therefore

$$p_w = \tfrac{1}{2}(0.85) + \tfrac{1}{2}(0.75) = 0.425 + 0.375 = 0.800, \qquad m_w = 2(0.800) - 1 = 0.600.$$

The displayed fixed inverse-peer statistics are $C_1 = 0.62$, $C_0 = 0.38$, and $\gamma = 0.5$. Every correction step is therefore

$$\begin{aligned} C_1 - C_0 &= 0.62 - 0.38 = 0.24, \\ c &= 0.5(0.24) = 0.12, \\ \Pi_{0.60}(0.12) &= \min\{0.12, 0.60 - \epsilon\} = 0.12 \end{aligned}$$

for the protocol value $\epsilon < 0.48$. Substitution gives

$$p_{\mathrm{SISP}} = \frac{1 + 0.60 - 0.12}{2} = \frac{1.48}{2} = 0.74 > 0.5.$$

The calculation therefore preserves the vulnerable decision. The preceding contract analysis independently supplies the concrete witness allowance write→untrusted callback →reentrant transferFrom→asset effect. The numerical trace documents the generic S-ISP computation shown in Figure 1; category-specific deployed supports and strengths are recorded separately in Appendix E.3.

E Reproducibility and Baseline Implementations

E.1 Corpus partitions and held-out evaluation

The three corpora remain separate throughout metric computation. Top200+DeFi uses immutable, label-balanced category partitions. FORGE assigns source projects, rather than individual files, to disjoint partitions and retains an exact source duplicate at most once per project. SmartBugs-v3 uses fixed, content-hashed single-file partitions, curated vulnerability annotations, and comment-stripped inputs. Together, they provide real-contract, project-disjoint, and cross-taxonomy evidence.

Table 14: Corpus sample information in contract–vulnerability pairs.

Corpus	Families	Reported scope	Labeled pairs
Top200+DeFi	3	Dataset	162
FORGE	3	Held-out evaluation	120
SmartBugs-v3	8	Held-out evaluation	248

Each record carries its source revision, label, split assignment, and SHA-256 digest. Training records supply rule evolution; development records determine support, weights, correction strength, and fitted aggregation parameters; and final-evaluation records are scored only after these choices are fixed. The primary-corpus audit confirms that deployed rules draw exclusively from training. Detector prompts contain normalized source and the vulnerability criterion, excluding labels, file names, and finding descriptions.

Within each corpus, category metrics are computed on fixed final-evaluation records. Corpus macros are unweighted over their reported families, and the all-family macro is unweighted over the 14 family rows in Table 1; it therefore measures equal family coverage rather than contract-weighted performance.

The evaluation record includes immutable split manifests, resolved settings, prompt/response hashes, terminal status, and contract-level predictions. Greedy decoding is used except for documented stochastic baseline stages and the explicit sampling ablation, which uses temperature 0.6, top-p 0.95, and three fixed seeds. Before tabulation, the audit checks final-split coverage, the expected observations per contract, successful completion, and agreement between resolved settings and the recorded source digest. Metrics are aggregated only after these checks pass.

E.2 Common-backbone baseline interface

The evaluated baselines are Detect-Llama Douglas et al. (2024), Detection Made Easy/SmartVD Gao et al. (2024), Logic Meets Magic Ma et al. (2025), GPTLens Hu et al. (2023), LLM-SmartAudit Wei et al. (2024), SmartLLM Kevin and Yugopuspito (2025), and SimilarGPT Chen et al. (2024). All use the shared backbone, the same normalized contract inputs, and their documented inference and decision procedures. Detect-Llama, Detection Made Easy, and Logic Meets Magic retain their vulnerability-specialized single-model reasoning procedures. GPTLens retains its auditor–critic stages, LLM-SmartAudit its role-based auditors, SmartLLM its staged generative workflow, and SimilarGPT its code-similarity evidence. Every adapter returns a score oriented as $P(\mathrm{vulnerable})$ and applies the common 0.5 decision boundary. The shared model, decoding policy, input text, threshold, and category labels make the main matrix a direct comparison of these cited detection mechanisms.

Baseline fidelity follows a two-level reproduction protocol. For methods with executable public source, the released pipeline is invoked directly on the fixed benchmark splits; only the dataset reader and model-service interface are connected to the normalized inputs and shared backbone. The released prompt roles, stage ordering, intermediate evidence flow, output parser, and decision rule are retained. If source is unavailable or cannot be executed in the common-backbone setting, we reconstruct the method from its paper and supplementary material, preserving its named technical mechanism rather than reducing it to a generic prompt. Each reconstruction is checked against a method-specific fidelity record covering required inputs, inference stages, requested calls, score semantics, parser behavior, and final decision rule. These checks establish that the comparison

changes the evaluation data and common backbone interface while retaining the operative procedure of each baseline.

The comparison retains each method's documented number and ordering of inference stages; measured calls and GPU-hours are therefore reported alongside performance. GPTLens retains its stochastic auditor at temperature 0.7 and a greedy critic. The stochastic auditor produces five samples per contract, and the corresponding auditor–critic predictions are averaged before metric computation. The samples are generated as one batched auditor-stage invocation, so the call-count accounting treats the batch as one stage call while measured GPU-hours include all generated sequences. The other main-table baseline runs use greedy decoding. Inputs, backbone, and the final decision boundary remain shared. Before table assembly, every adapter is validated for requested-category emission, score orientation, and configured-threshold propagation. Detect-Llama and SmartLLM consequently use the requested vulnerability category and the common 0.5 boundary throughout. When an immutable aggregate record already contains the requested per-contract scores, its category view is regenerated with the source digest preserved; when category parsing affects the stored response, inference is repeated. The validation suite covers category normalization, benign-score orientation, and threshold propagation. No final-evaluation label is used to alter an adapter, threshold, or score orientation.

E.3 Aggregation controls and resolved operating point

The aggregation study also includes two capacity comparators using the same four cached auditor scores: logistic stacking learns main effects, whereas quadratic stacking adds every $s_i s_j$ interaction. Both are fit only on development examples, providing a favorable test of whether generic pairwise capacity, rather than S-ISP's inverse-state structure, explains the gain.

Four-auditor bilateral S-ISP selected strengths 0.5, 0.25, and 0 for the three categories. The primary protocol fixes the a priori 0.5 boundary for every reported decision; category-specific development scans are diagnostics only.

For sparse deployment, the selector enumerates the six unordered two-auditor supports, uniform and reliability weighting, and $\Gamma = \{0, 0.25, 0.5, 0.75, 1.0, 1.5\}$. Every leave-one-out development prediction refits the applicable reliability and pairwise quantities after excluding the scored item. The deterministic objective in Eq. 1 first maximizes LOO F1, then applies the prespecified complexity preference, Brier tie-break, lexical support order, and correction strength order. The selected tuple is serialized before final-evaluation metrics are computed.

The development-only selected sparse supports (one-based) are $(2, 3)$, $(2, 4)$, and $(1, 2)$ for logic error, access control, and reentrancy, with selected strengths 0.5, 1.0, and 0. In the same order, α_v is 2.0, 4.0, and 0.5; $\lambda = 10^{-3}$ and $\epsilon = 10^{-9}$. All values are fixed before held-out prediction.

The auditor-budget record permits support sizes up to $k \in \{1, 2, 3, 4\}$, separating information availability from forced deployment; paired significance and the complete three-seed sampling summary are reported above.