SR-Fraud: An Outcome-Supervised Reflective LLM Agent Framework for Non-Stationary Payment Fraud Detection

Xuwei Tan *
Coinbase, Inc.
The Ohio State University

Yao Ma
Coinbase, Inc.

Xueru Zhang
The Ohio State University

arXiv:2609.27287v1 [cs.LG] 23 Sep 2026

Abstract

Real-time payment fraud detection is a non-stationary streaming prediction problem: adversaries adapt before supervised labels mature, and localized burst attacks can cause losses before retraining. Production systems typically rely on tabular classifiers and rules, which can struggle to capture these emerging sequential patterns before periodic retraining occurs. We present **SR-Fraud**, an outcome-supervised reflective LLM framework that decouples request-time decisions from offline adaptation. A frozen, stateless agent scores each transaction from a Hybrid Episodic Window to track behavioral shifts, while an offline reflection agent proposes boundary hypotheses from matured errors. A deterministic verifier then admits only supported hypotheses into an executable knowledge state. On a production payment-fraud benchmark, SR-Fraud improves all detection metrics over its frozen decision agent, obtains higher point estimates than static and periodically retrained CatBoost, and detects an emerging fraud burst.

1 Introduction

Payment fraud in financial platforms is an inherently non-stationary streaming problem (Lu et al., 2018; Dal Pozzolo et al., 2015). As adversaries continuously adapt to bypass active defenses, fraud patterns learned in static training data quickly decay. This non-stationarity is particularly challenging during localized burst attacks, where adversaries execute high-value operations in a short window before a model can adapt to the new pattern.

Production pipelines typically rely on tabular classifiers like Gradient Boosted Decision Trees (GBDTs) (Chen and Guestrin, 2016; Prokhorenkova et al., 2018) and rule engines (Soui et al., 2019). However, they face two limitations. First, fraudulent transactions are often individually plausible, becoming recognizable only when evaluated against historical user behavior. Standard GBDTs score rows independently unless temporal behavior is explicitly encoded through engineered aggregate features (Bahnsen et al., 2016). Second, responding to distribution shifts requires periodic retraining and newly engineered features.

Large Language Models (LLMs) offer a complementary paradigm for non-stationary risk streams. Utilizing pre-trained world knowledge, LLMs naturally comprehend complex, high-cardinality metadata zero-shot (e.g., recognizing disposable email domains as potential risk signals), without manual feature engineering or model training. In addition, by comparing current transactions against a user's recent behavioral history via in-context learning (Brown et al., 2020), LLMs intuitively uncover subtle anomalies and effectively handle transient fraud spikes. Finally, the explicit natural-language insights and rules synthesized by the LLM can be transferred across diverse product lines.

Despite these benefits, directly integrating generic LLM agents into inline fraud pipelines remains non-trivial due to two core bottlenecks. On one hand, real-time payment systems are governed by strict latency budgets. Advanced inference-time deliberation (Shinn et al., 2023; Madaan et al., 2023; Yao et al., 2023a), while effective, induces unacceptable latency overhead, leading to a prominent **reasoning-budget gap**. On the other hand, deployed LLMs are frozen, meaning their general knowledge cannot natively adjust to dynamic data patterns, highly institution-specific fraud definitions, or operational policies. This causes a **adaptation gap** between pre-trained general capabilities and the need for fine-grained domain alignment.

To bridge these gaps, we introduce **SR-Fraud**, an outcome-supervised reflective LLM agent framework that decouples real-time inference from

*Work done during internship at Coinbase. This paper contains the author's personal opinions and does not constitute a company policy or statement. These opinions are not endorsed by or affiliated with Coinbase, Inc. or its subsidiaries.

asynchronous offline learning. At request time, a stateless decision agent scores each transaction in a single pass over a **Hybrid Episodic Window**, combining recent user sequences with localized velocity signals. This ensures the model reasons over behavioral changes rather than isolated events, while staying within latency budgets. Offline, an **Outcome-Supervised Reflection Agent** analyzes matured errors[1] to propose new rule-based hypotheses. Rather than relying on LLM self-critique, a deterministic harness verifies these proposals against historical labels. Verified rules accumulate into an **Evolving Expert Knowledge State**, a compact and human-readable set of rules that adjusts the decision agent's boundary cases. Our key contributions are summarized as follows:

- We introduce **SR-Fraud**, an architecture that keeps offline reflection off the request path and requires only *one* decision-model invocation per reviewed transaction, bridging the *reasoning-budget gap* in production pipelines.
- We propose an **outcome-supervised reflection mechanism** that forms a compact symbolic knowledge state through propose-and-verify hypotheses on the decision agent's outcomes, closing the *adaptation gap*.
- We benchmark existing LLM-based methods and SR-Fraud on **data from our production stream**, demonstrating the behavior of different LLMs and the benefits of SR-Fraud relative to traditional tabular classifiers.
- We further explore **fraud-domain LLM post-training** and analyze deployment trade-offs between token-priced commercial APIs and self-hosted open-source models.

2 Methodology

2.1 System Overview

In production deployments, risk systems process transactions through a standard machine learning model (e.g., GBDTs) with rule engines performing broad, low-latency screening, while SR-Fraud is selectively invoked on those high-risk or ambiguous edge cases requiring complex contextual reasoning. This modular design balances latency and compute costs while preserving the high-throughput guarantees of the primary risk pipeline. In this work, we focus specifically on the LLM agent side and omit the upstream model contents. Architecturally, SR-Fraud enforces a separation between request-time inference and asynchronous offline learning. A frozen, stateless *decision agent* assigns each transaction one of five bounded risk levels from a Hybrid Episodic Window. The resulting boundary scores are then adjusted by an *Evolving Expert Knowledge State*, a compact symbolic store of verified corrections maintained by the *reflection agent*. Figure 1 shows the framework.

[1]Maturation refers to the delay before true transaction labels are confirmed, typically via user-filed chargebacks.

2.2 Transaction-Grounded Decision Agent

At request time, SR-Fraud acts on a payment-risk stream. Given an incoming transaction x_t from user u, the decision agent must assign a risk level before the payment settles. A transaction that appears plausible in isolation may nevertheless depart sharply from the user's established behavior. We therefore cast scoring as contextual anomaly detection rather than single-record classification.

Let $R_u(t) = (x_{t_1}, \ldots, x_{t_m})$ be a bounded, time-ordered window of user u's m transactions preceding time t. SR-Fraud formats this event stream as a *Hybrid Episodic Window*,

$$W_u(t) = \big(R_u(t), V_u(t)\big), \tag{1}$$

where $V_u(t)$ is a burst-and-velocity summary computed from $R_u(t)$. The decision context is

$$c_t = \Phi\big(x_t, W_u(t)\big), \tag{2}$$

where Φ serializes the evidence as structured text. The LLM maps this context to a bounded risk level $r(x_t) \in \{1, \ldots, 5\}$. We take $r(x_t) \geq 4$ as a block.

The Hybrid Episodic Window makes temporal comparison the agent's core reasoning primitive. The sequence $R_u(t)$ retains operational attributes such as payment rail, amount, and destination while excluding personally identifiable information. The summary $V_u(t)$ complements the raw sequence with retry, timing, amount, and channel-velocity signals. Presenting both views preserves event-level context while making short bursts explicit.

The score is then further corrected by an *Evolving Expert Knowledge State* K, which is maintained and updated by the *reflection agent* (§2.3). After the agent emits $r(x_t)$, K edits only the boundary cases: *escalating* an under-blocked $r{=}3$ transaction to a block, or *releasing* an over-blocked $r{=}4$ transaction to an allow. All adaptation therefore occurs through a deterministic boundary layer, leaving the decision agent stateless and single-pass.

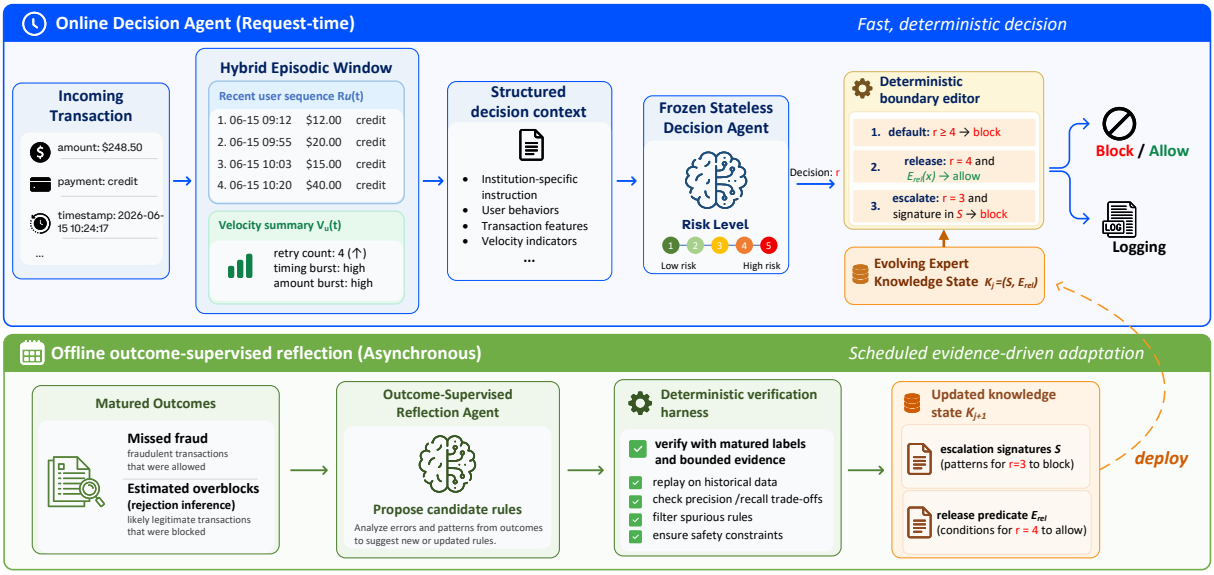


Figure 1: Overview of SR-Fraud. At request time, a fast stateless agent assigns a bounded risk level from the current transaction and Hybrid Episodic Window, after which the expert knowledge state deterministically adjusts boundary decisions. Offline reflection uses matured missed fraud and estimated overblocks to update K for the next cycle.

2.3 Outcome-Supervised Reflection

Outcome-supervised reflection converts matured payment outcomes into actionable updates for the *Evolving Expert Knowledge State*. Fraud labels become reliable only after real-world outcomes, such as chargebacks or investigations, mature. Once per weekly cycle, the reflection agent analyzes the available resolved errors and proposes candidate rules. A deterministic harness verifies these candidates against matured data. Thus, the LLM directs *where to search*, while evidence determines *what enters the active policy*. We use t for transaction time and j for the weekly deployment cycle. Within cycle j, the state K_j is fixed for all request-time decisions.

Boundary-Editing Expert Knowledge State: Reflection avoids rewriting the underlying LLM policy, mitigating the risk of overturning high-confidence decisions. Given the frozen agent's score $r(x) \in \{1, \ldots, 5\}$ where $r(x) \geq 4$ defaults to a block, the state K edits only boundary cases: escalating suspicious $r(x) = 3$ transactions and releasing over-blocked $r(x) = 4$ transactions.

Suppressing the cycle index, write the active state as $K = (\mathcal{S}, E_{\text{rel}})$, where $\mathcal{S}$ is a set of conjunctive escalation signatures and E_{rel} is a Boolean release predicate. We write $K_\varnothing = (\varnothing, \bot)$ for the empty state, where $\bot$ is the always-false release predicate. The final deterministic decision policy is below, where $[\cdot]$ denotes an Iverson indicator:

$$\begin{aligned} \text{esc}(x) &= [r(x){=}3] \wedge \textstyle\bigvee_{s \in \mathcal{S}} s(x), \\ \text{rel}(x) &= [r(x){=}4] \wedge E_{\text{rel}}(x), \\ \text{block}(x) &= \big([r(x){\geq}4] \vee \text{esc}(x)\big) \wedge \neg\, \text{rel}(x). \end{aligned} \tag{3}$$

Assuming a fixed-size release predicate, the boundary layer executes in $O(|\mathcal{S}|)$ time, and $K = K_\varnothing$ exactly reproduces the decision agent. Every $s \in \mathcal{S}$ is a conjunction over the leaf-feature vocabulary, as is the release predicate E_{rel}. The set $\mathcal{S}$ is grown and pruned across cycles, whereas E_{rel} is a single release condition.

Reflection Cycle and Selective Labels. To dynamically update this state K, the reflection agent analyzes the online policy's matured errors at each deployment cycle j. However, acquiring these errors faces a selective-labels asymmetry: missed fraud is easily observable from settled transactions, but false blocks censor their own counterfactual outcomes. Consequently, we compile two matured pools entirely from observable, allowed transactions from previous cycles: the missed-fraud pool ($\mathcal{F}_j$) to drive escalation mining, and the observable $r{=}3$ bank ($\mathcal{B}_{3,j}$) to ground release safety via *rejection inference*. A legitimate-dense band in settled $\mathcal{B}_{3,j}$ traffic proxies the censored $r{=}4$ traffic, estimating the true positives a release would surrender without relying on unobserved block labels; we validate this proxy against an oracle in Appendix D.7.

The harness renders this evidence into a tex-

tual digest. The offline LLM proposes $\mathcal{G}_j = (\mathcal{A}_j, \mathcal{Q}_j, \mathcal{L}_j)$, where $\mathcal{A}_j$ contains amount atoms, $\mathcal{Q}_j$ contains qualifier atoms, and $\mathcal{L}_j$ contains candidate release bands. The deterministic harness enumerates the escalation language $\mathcal{H}_j = \{a \wedge q_1[\wedge q_2] : a \in \mathcal{A}_j,\ q_1, q_2 \in \mathcal{Q}_j\}$, where $[\wedge q_2]$ denotes an optional second qualifier. It then verifies the candidates and updates K. Thus, the LLM defines the search space, while the harness controls validation (Algorithm 1).

Memory Evolution. Because deployed rules block matching transactions, their counterfactual outcomes are no longer observed. We therefore re-evaluate all signatures each cycle. To avoid pruning a pre-test-verified rule on a thin matured window, we retire it only when its marginal-precision Wilson *upper* bound is below the operating threshold and it also fails the original train–validation gate. Candidates that clear the gate are ranked by their Wilson lower bound and admitted up to a fixed per-cycle cap. The knowledge state is updated once per cycle from matured evidence; its initialization and replay trajectory are detailed in Appendix B.

Verification by Bounded Evidence. To mitigate the winner's curse when evaluating many candidates on short time windows, the harness uses one-sided Wilson confidence bounds. Suppressing the cycle index j, let p_{op} be the decision agent's operating precision, $p_{\text{marg}}(h)$ the precision of transactions newly blocked by escalation h, and $p_{\text{free}}(b)$ the fraud rate among transactions released by band b. We require:

$$\begin{aligned} \text{LCB}_{\text{Wil}}(p_{\text{marg}}(h)) &> \gamma\, p_{\text{op}}, \\ \text{UCB}_{\text{Wil}}(p_{\text{free}}(b)) &< p_{\text{op}}. \end{aligned} \tag{4}$$

Together with structural pre-tests, these bounds constrain boundary updates.

3 Experiments

3.1 Data and Replay Protocol

Dataset. We evaluate a fixed cohort of 16,140 transactions from nine weeks of a production payment-risk stream with an overall 5.04% fraud rate (Table 1); it is a case–control sample (all matured fraud, subsampled legitimate traffic), and every system is scored on all 16,140 transactions. Training and validation each cover two preceding months.[2] Appendix C.1 provides construction details.

[2]Company policy limits disclosure of the broader stream's distribution and sampling statistics.

Statistic	Benchmark
Transactions	16,140
Fraud transactions	813
Fraud rate	5.04%
Entities (P / N)	7,293 (353 / 6,940)
Fraud per week	48/86/73/38/73/67/90/**235**/103

Table 1: Evaluation benchmark. P/N denotes entities with/without confirmed fraud; per-week fraud counts reveal a burst in week 8.

Replay protocol. We use time-ordered splits and out-of-time evaluation rather than random partitions. The validation period is used for model selection, and the final period is replayed in timestamp order. Earlier training and validation events may appear as prior transaction history, but their labels are never exposed to the decision agent.

Metrics. We evaluate precision, recall, F1, and **dollar-weighted recall**, defined as the ratio of the total amount value of correctly blocked fraudulent transactions to the total value of all actual fraud.

3.2 Implementation Details

We replay the test stream in weekly cycles with a fixed two-cycle maturation lag: the policy for cycle j may use outcomes only through cycle $j-2$. This maturation wall is enforced by the replay harness, and every cycle is scored once and then frozen, without retrospective re-scoring under a later knowledge state. Before test replay, the same propose–verify pipeline is applied to the preceding train–validation period; subsequent updates use only outcomes exposed by the maturation wall. Full implementation details are provided in Appendix B.

3.3 Comparison Systems and Baselines

We compare systems that isolate tabular learning, the LLM prior, behavioral history, and outcome-supervised adaptation.

- **CatBoost.** It is trained on the full training period, with hyperparameter optimization using Optuna. We also evaluate **CatBoost-periodical**, the same model periodically retrained over the stream.
- **Seq-GRU.** A supervised GRU over the last-k transaction sequence as the decision agent (Jurgovsky et al., 2018).
- **Direct LLM.** The LLM scores only the current transaction (Hegselmann et al., 2023).

Method	ΔPrec.	ΔRecall	ΔF1	ΔRecall$	Block	p_{90} (s)	Mean (s)
CatB. (ref.)	0.0	0.0	0.0	0.0	5.00%	<0.1	<0.1
CatBoost-periodical	+4.2	+4.1	+4.2	+3.9	5.00%	<0.1	<0.1
Seq-GRU	-2.5	+6.5	+1.4	+4.0	5.00%	<0.1	<0.1
Direct LLM (current transaction only)							
GPT-5.4	-18.5	-4.0	-13.7	-15.5	10.32%	1.06	0.81
Opus 4.5	-19.2	-7.1	-15.0	-18.0	9.68%	2.01	1.35
Sonnet 4.6	-8.0	-21.6	-17.5	-35.7	2.16%	1.71	1.21
Decision Agent-only: SR-Fraud online agent, no reflection							
GPT-5.4	-11.8	+2.8	-6.4	-11.0	8.60%	1.70	1.27
Opus 4.5	-2.1	-1.5	-1.8	-15.3	5.10%	1.97	1.28
Sonnet 4.6	-0.9	+5.0	+1.8	-7.6	5.97%	2.02	1.44
Reflexion: verbal reflection (risk ≥ 4)							
GPT-5.4	-25.1	+25.4	-19.6	+10.8	41.44%	2.48	2.22
Opus 4.5	-22.2	+31.7	-14.8	+19.5	32.24%	2.81	2.00
Sonnet 4.6	-13.3	+23.5	-3.9	+26.1	14.80%	2.85	1.70
SR-Fraud (full system)							
GPT-5.4	-1.2	+7.6	+2.7	+2.8	6.44%	1.71	1.28
Opus 4.5	**+6.2**	+3.0	+4.6	-2.2	4.59%	1.97	1.28
Sonnet 4.6	+4.5	**+12.3**	**+8.0**	**+11.7**	6.08%	2.03	1.45

Table 2: Overall performance as percentage-point differences from static CatBoost. Block rates and latency remain absolute.

- **Decision Agent-only.** The SR-Fraud decision agent without reflection or an evolving knowledge state.
- **Reflexion.** A weekly verbal-memory baseline adapted from Shinn et al. (2023).
- **SR-Fraud.** The decision agent augmented with our outcome-supervised reflection mechanism.

Full feature, training, and adaptation details for all baselines are provided in Appendix C.2.

3.4 Results and Analysis

3.4.1 Overall Detection Performance

Table 2 yields three main findings.

Fraud is hard to discover from a single transaction, yet LLMs are strong at reasoning over behavior. Relative to static CatBoost, the direct Sonnet 4.6 reviewer has deltas of -8.0 precision, -21.6 recall, -17.5 F1, and -35.7 dollar-weighted recall. Adding recent history changes these gaps to -0.9, $+5.0$, $+1.8$, and -7.6, respectively. Opus and GPT-5.4 show the same direction, indicating that behavioral comparison supplies the main signal.

Reflection improves precision, recall, and dollar-weighted recall together. Relative to the Sonnet 4.6 decision agent, full SR-Fraud improves precision by $+5.4$, recall by $+7.3$, F1 by $+6.2$, and dollar-weighted recall by $+19.3$ points, at essentially the same block rate ($5.97\% \rightarrow 6.08\%$). All four paired bootstrap intervals exclude zero, including $+6.2$ F1 $[3.6, 9.0]$ and $+19.3$ Recall$ $[14.3, 24.5]$. Against CatBoost-periodical at its 5% point, the deltas are $+0.3$ precision, $+8.1$ recall, $+3.8$ F1, and $+7.8$ dollar recall; recall and dollar recall exclude zero, while precision and F1 do not. On cohorts with at least one prior transaction, the recall, F1, and dollar-recall advantages over CatBoost-periodical are significant; relative to the decision agent, the dollar-recall gain remains significant in the ≥ 1 and ≥ 2 cohorts (Appendix D.1).

The larger dollar-recall gain is consistent with feedback asymmetry: missed fraud provides direct labels for escalation, whereas releases rely on a conservative proxy for censored blocks. This favors recovery of value-skewed missed fraud.

Free-text reflection can destabilize the operating point. Reflexion results show that purely verbal reflection, which summarizes past outcomes in a free-text memory, can accumulate overly broad or aggressive patterns and cause the LLM to block more legitimate transactions. SR-Fraud instead converts reflection into structured candidate rules and verifies each rule against matured evidence before admission. This shows that unverified free-text reflection can be unstable at the required operating point.

3.4.2 Behavioral Context Drives LLM

Table 3 evaluates fixed models across cumulative history cohorts. As history deepens, the LLM advantage over both CatBoost variants widens with

Prior history	n	ΔPrecision				ΔDollar-recall			
		CatB.	CatB. period.	Dec. agent	SR-Fraud	CatB.	CatB. period.	Dec. agent	SR-Fraud
≥ 0 (all)	16,140	0.0	+4.2	-0.9	**+4.5**	0.0	+3.9	-7.6	**+11.7**
≥ 1 prior	10,263	0.0	+5.8	+7.5	**+9.5**	0.0	+7.9	+9.9	**+22.3**
≥ 2 prior	7,391	0.0	+8.6	+15.0	**+15.0**	0.0	+11.4	+27.1	**+35.0**
≥ 3 prior	5,650	0.0	+11.9	+21.2	**+21.8**	0.0	+13.6	+33.1	**+43.5**

Table 3: Performance by prior-history depth as percentage-point differences from static CatBoost within each cohort.

Detector	ΔRecall	$\Delta\text{Recall}_\$$
Week-8 burst ($n = 206$)		
CatBoost (reference)	0.0	0.0
CatBoost-periodical	+9.2	+6.6
Decision agent (Sonnet 4.6)	+38.4	+34.8
SR-Fraud	**+41.7**	**+46.4**
Emerging high-value vector ($n = 55$)		
CatBoost (reference)	0.0	0.0
CatBoost-periodical	-3.7	-1.9
Decision agent (Sonnet 4.6)	-16.4	-15.3
SR-Fraud	**+10.9**	**+10.4**

Table 4: Burst and high-value performance as percentage-point differences from static CatBoost within each subset. Other systems retain their native operating points.

history depth. Although CatBoost includes rolling history aggregates, these fixed summaries may encode a broad prior that entities with longer histories are safer, while the LLM instead compares the current event directly with the raw episodic sequence. The widening gap motivates further routing history-rich traffic to the contextual LLM layer.

3.4.3 Detection of an Emerging Fraud Burst

We isolate the largest test-stream burst, a coordinated low-value fraud attack comprising 206 of the 235 confirmed frauds in week 8, and separately examine a predefined high-value fraud subset (55 frauds) where prevented loss is concentrated (Table 4). The subset definition was fixed before evaluating model performance; its numerical cutoff is withheld under internal disclosure policy.

The base LLM detects the unseen burst zero-shot. Both CatBoost and CatBoost-periodical miss much of the burst because their latest models predate the emerging pattern. At their native operating points, the LLM identifies the burst from recent transaction context without first observing labeled examples of the attack; Appendix D.5 confirms the comparison at an exact 6% budget.

Reflection complements burst detection with high-value coverage. On the burst, SR-Fraud exceeds static CatBoost at its native 5% point by $+41.7$ recall points and $+46.4$ dollar-recall points, compared with $+38.4$ and $+34.8$ for the decision agent. On the high-value subset, SR-Fraud's deltas are $+10.9$ and $+10.4$, whereas the decision agent's are negative. Under the two-cycle maturation wall, week-8 outcomes become available only after the replay ends, so reflection cannot have learned from this burst itself. The two components therefore play distinct roles: the base LLM detects the new volume burst from behavioral context, while previously verified boundary rules found on earlier matured cycles improve high-value coverage.

3.4.4 Post-Training the Decision Agent

While the central contribution of this work is the reflection framework, we additionally explore post-training the decision agent itself. We include this study because its performance, cost, and latency implications are of interest to industry practitioners deploying LLM risk agents at scale; concretely, we compare a commercial API backbone against a self-hosted, domain-adapted model along these three axes.

On cost and latency, commercial APIs reduce infrastructure work but introduce per-token pricing, variable network latency, and limited control over model upgrades; and on performance, newer Opus, Sonnet, and GPT releases in fact regress on this benchmark (Table 20). At sustained traffic volume, self-hosting instead trades per-request charges for predictable GPU capacity while enabling domain-specific training and tighter data control. We therefore post-train a self-hosted 9B decision agent using outcome-supervised rewards. Despite its smaller size and lack of a symbolic knowledge state, SR-Fraud-PT attains similar detection performance while reducing request latency and, at sufficient sustained volume, modeled serving cost. Table 5 reports the result; Appendix D.12

System	ΔP	ΔR	ΔF1	$\Delta R_{\$}$	p_{90} (s)
CatBoost (reference)	0.0	0.0	0.0	0.0	<0.1
Decision Agent	-0.9	+5.0	+1.8	-7.6	2.02
SR-Fraud	+4.5	+12.3	+8.0	+11.7	2.03
Base 9B (no post-train)	-18.8	-9.1	-15.2	-25.0	0.94
SR-Fraud-PT (9B)	+5.4	+10.5	+7.8	+20.1	0.94

Table 5: Commercial and self-hosted systems as percentage-point differences from static CatBoost. Latency remains absolute.

gives the training recipe and latency measurement.

4 Conclusion

We presented **SR-Fraud**, which combines a stateless contextual LLM with outcome-supervised symbolic reflection for non-stationary payment fraud detection. A deterministic verifier admits and retires boundary rules from matured evidence, preserving one request-time LLM call and an auditable knowledge state. Chronological replay shows that behavioral context and verified reflection improve fraud capture over static and periodically retrained tree baselines; an exact block-count comparison preserves the advantage while holding customer friction fixed.

Limitations

Confidence bounds. The verifier uses a heuristic one-sided Wilson bound without a formal multiple-testing correction. Re-verifying the persisted main-run candidates across the tested confidence settings yields a stable operating point (Appendix D.8); this sensitivity check does not, however, constitute formal control of adaptive multiple comparisons.

Hosted-model stochasticity. Performance is stable across the evaluated reflection/decision-backbone configurations. Repeated full-stream decision replays were not conducted because of their API and computational cost. The entity bootstrap quantifies sampling uncertainty.

Proprietary evaluation and generality. The study uses one non-releasable production stream. Public fraud benchmarks are typically synthetic or highly anonymized and do not preserve the metadata, entity histories, and delayed-label dynamics required here. The production-derived cohort improves realism but limits independent replication and cross-platform generalization. Company policy also precludes disclosure of the legitimate sampling rate, upstream routing logic, represented traffic share, and production prevalence; we therefore make no end-to-end production-rate claim. Metrics are reported as percentage-point differences relative to the corresponding CatBoost reference.

Fixed maturation lag. The two-cycle delay approximates the dominant label-maturation pattern summarized from the historical production stream. The replay uses this fixed wall rather than transaction-specific availability times, providing a reproducible approximation while abstracting away proprietary timing details.

Hosted-provider latency. The reported hosted percentiles are point-in-time, request-level measurements through model providers for this benchmark, rather than production SLO guarantees or full application latency. Provider-side load, request routing, service changes, and network conditions can change these values.

Off-policy replay. The replay applies SR-Fraud's decisions without altering the stream, so a transaction the policy would have blocked still appears in later histories. Confirmed fraud usually triggers entity denylisting, making subsequent same-entity fraud rare after confirmation. This reduces the issue: pre-confirmation attempts and legitimate counterfactual blocks could still reshape histories. Policy-consistent or shadow evaluation is left to future work.

Ethics Statement

The production system operates on payment transaction data. Direct identifiers and designated PII fields are excluded from model prompts. Our reported experiments run the reflection loop fully automatically; in production deployment, disputes are reviewed by a human analyst as part of normal operations and the reflection-loop knowledge update passes an optional human-review checkpoint before deployment.

Only a vetted subset of transaction attributes enters the prompt, and content is serialized as data rather than instructions. These controls reduce, but do not eliminate, metadata, re-identification, or prompt-injection risk.

References

Anthropic. 2025. Introducing Claude Opus 4.5.

Anthropic. 2026. Claude Sonnet 4.6 System Card.

Alejandro Correa Bahnsen, Djamila Aouada, Aleksandar Stojanovic, and Björn Ottersten. 2016. Feature engineering strategies for credit card fraud detection. *Expert Systems with Applications*, 51:134–142.

Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J Christopher Westland. 2011. Data mining for credit card fraud: A comparative study. *Decision support systems*, 50(3):602–613.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Shaosheng Cao, XinXing Yang, Cen Chen, Jun Zhou, Xiaolong Li, and Yuan Qi. 2019. Titant: online real-time transaction fraud detection in ant financial. *Proceedings of the VLDB Endowment*, 12(12):2082–2093.

Fabrizio Carcillo, Andrea Dal Pozzolo, Yann-Aël Le Borgne, Olivier Caelen, Yannis Mazzer, and Gianluca Bontempi. 2018. Scarff: a scalable framework for streaming credit card fraud detection with spark. *Information fusion*, 41:182–194.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

William W Cohen. 1995. Fast effective rule induction. In *Machine learning proceedings 1995*, pages 115–123. Elsevier.

Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. 2015. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *2015 international joint conference on Neural networks (IJCNN)*, pages 1–8. IEEE.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, and 1 others. 2024. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pages 57734–57811.

Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12.

Yufei He, Ruoyu Li, Alex Chen, Yue Liu, Yulin Chen, Yuan Sui, Cheng Chen, Yi Zhu, Luca Luo, Frank Yang, and Bryan Hooi. 2025. Enabling self-improving agents to learn at test time with human-in-the-loop guidance. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1625–1653, Suzhou (China). Association for Computational Linguistics.

Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. 2023. TabLLM: Few-shot classification of tabular data with large language models. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, pages 5549–5581. PMLR.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2023. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Johannes Jurgovsky, Michael Granitzer, Konstantin Ziegler, Sylvie Calabretto, Pierre-Edouard Portier, Liyun He-Guelton, and Olivier Caelen. 2018. Sequence classification for credit-card fraud detection. *Expert systems with applications*, 100:234–245.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, 31(12):2346–2363.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594.

OpenAI. 2026. GPT-5.4 Thinking System Card.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31.

Bo Qu, Zhurong Wang, Daisuke Yagi, Zach Xu, Yang Zhao, Yinan Shan, and Frank Zahradnik. 2025. Llm-enhanced self-evolving reinforcement learning for multi-step e-commerce payment fraud risk detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 92–103.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Makram Soui, Ines Gasmi, Salima Smiti, and Khaled Ghédira. 2019. Rule-based credit risk assessment model using multi-objective evolutionary algorithms. *Expert systems with applications*, 126:144–157.

Xuwei Tan, Yao Ma, and Xueru Zhang. 2026. Understanding structured financial data with llms: A case study on fraud detection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23375–23390.

Ruiyu Wang, Zifeng Wang, and Jimeng Sun. 2023. Unipredict: Large language models are universal tabular classifiers. *arXiv preprint arXiv:2310.03266*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Yi Yang, Mark Christopher Siy Uy, and Allen Huang. 2020. Finbert: A pretrained language model for financial communications. *arXiv preprint arXiv:2006.08097*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

A Related Work

Payment fraud detection. Payment fraud detection operates on heterogeneous transaction data under evolving behavior and delayed supervision. Prior work compares data-mining classifiers (Bhattacharyya et al., 2011), studies temporal feature engineering (Bahnsen et al., 2016), and analyzes concept-drift adaptation generally (Gama et al., 2014) and with delayed fraud labels specifically (Dal Pozzolo et al., 2015). Sequence-based models directly exploit ordered card activity (Jurgovsky et al., 2018), while SCARFF studies scalable streaming credit-card fraud detection (Carcillo et al., 2018). TitAnt presents an online, real-time transaction-fraud system at Ant Financial (Cao et al., 2019). Scalable tree learners such as XGBoost, LightGBM, and CatBoost provide efficient tabular baselines (Chen and Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018). SR-Fraud is complementary to this literature: it retains the tabular first pass but adds contextual LLM reasoning and a verified symbolic adaptation layer.

LLMs and foundation models for financial and tabular data. TabLLM studies few-shot classification by representing tabular examples as text (Hegselmann et al., 2023), while UniPredict investigates LLMs as general tabular classifiers (Wang et al., 2023). TabPFN provides a complementary foundation-model approach specialized for small tabular classification problems (Hollmann et al., 2023). In finance, FinBERT targets financial text (Yang et al., 2020), and BloombergGPT studies domain-specific language-model pretraining for finance (Wu et al., 2023). In fraud-specific settings, Tan et al. (2026) present a case study of LLMs on structured financial data, and Qu et al. (2025) study LLM-enhanced self-evolving reinforcement learning for multi-step e-commerce fraud detection. SR-Fraud instead focuses on a frozen request-time scorer and outcome-supervised symbolic updates, without online parameter optimization.

Reflective and self-improving agents. Prior agents improve behavior through verbal memory or iterative feedback: Reflexion stores task-derived reflections (Shinn et al., 2023), Self-Refine revises outputs using self-feedback (Madaan et al., 2023), and Tree of Thoughts searches intermediate reasoning states (Yao et al., 2023a). Related systems interleave reasoning and actions (Yao et al., 2023b), combine persistent memory with reflection (Park et al., 2023), use external-tool feedback (Gou et al., 2024), or request targeted human guidance to update a knowledge repository (He et al., 2025). SR-Fraud instead proposes hypotheses from weekly error batches and admits a boundary rule only after deterministic verification against matured labels; its replay requires no human guidance.

B SR-Fraud Implementation Details

B.1 Verification and Replay Configuration

Reflection Configuration. All reflection constants are fixed across cycles. The pre-test rarity ceiling is $\rho = 0.020$ of legitimate traffic. Its conditional-precision floor is 0.30 on train–validation, with at least 10 matched frauds. Hyperparameters are chosen based on the validation phase. Both one-sided Wilson bounds use $z = 1.0$, and the escalation admission margin is $\gamma = 1.15$. An escalation requires at least 8 marginal rows and 2 new true positives. For per-cycle corroboration, a cycle votes after 2 matched boundary rows, and a candidate needs at least 2 voting cycles. Retirement requires at least 8 marginal rows, a marginal Wilson upper bound below the operating floor, and failure of the original train–validation gate. A release band requires 20 settled boundary rows, permits at most one dissenting cycle, and must be exercised by settled data to at least half its proposed ceiling. Up to five escalation signatures that clear the gate may be admitted per cycle, ordered by their one-sided Wilson lower bound on marginal precision.

Temporal protocol. Cycles are weekly with a two-cycle maturation lag. To model production label delay conservatively and reproducibly, the replay does not reveal a cycle's outcomes until two later cycles, even though the offline benchmark physically contains every matured label. Reflection runs once per cycle as an offline agentic workflow and may use multiple model calls, none of which lies on the request path. The policy for cycle j is formed only from cycles at most $j-2$. Each cycle is scored once and frozen; aggregate results never rescore an earlier cycle under a later K. The empty state $K_\varnothing$ exactly reproduces the frozen decision agent. Although the backtest records physically contain settled labels for every cycle, the digest builder exposes only outcomes permitted by the maturation wall; this restriction is enforced by the harness rather than by agent behavior.

Algorithm 1 Automated reflection and deployment for cycle j.

Require: Matured data $\mathcal{D}_{\le j-2}$; prior state $K_{j-1} = (\mathcal{S}_{j-1}, E_{\mathrm{rel},j-1})$

$d_j \leftarrow \textsc{Digest}(\mathcal{D}_{\le j-2}, K_{j-1})$
$\mathcal{G}_j \leftarrow \textsc{ReflectAgent}(d_j)$, where $\mathcal{G}_j = (\mathcal{A}_j, \mathcal{Q}_j, \mathcal{L}_j)$
$\mathcal{H}_j \leftarrow \textsc{Enumerate}(\mathcal{A}_j, \mathcal{Q}_j)$
$(\mathcal{V}_j^{\mathrm{esc}}, \mathcal{V}_j^{\mathrm{rel}}) \leftarrow \textsc{Verify}(\mathcal{H}_j, \mathcal{L}_j, \mathcal{D}_{\le j-2}, K_{j-1})$
$E_{\mathrm{rel},j} \leftarrow \textsc{SelectRelease}(\mathcal{V}_j^{\mathrm{rel}}, E_{\mathrm{rel},j-1})$
$\widetilde{\mathcal{S}}_j \leftarrow \textsc{Retire}(\mathcal{S}_{j-1}, \mathcal{D}_{\le j-2}, E_{\mathrm{rel},j})$
$\mathcal{S}_j \leftarrow \textsc{AdmitBest}(\widetilde{\mathcal{S}}_j, \mathcal{V}_j^{\mathrm{esc}})$
$K_j \leftarrow (\mathcal{S}_j, E_{\mathrm{rel},j})$
$\textsc{DecideCycle}(j, K_j)$

Ensure: Updated state K_j

Pre-replay construction. Before test replay, SR-Fraud constructs an initial knowledge state by applying the same propose–verify procedure to the training and validation splits. The anonymized atom vocabulary is generated at this point. During replay, the weekly loop updates the deployed state using only matured test-stream outcomes. The complete replay trajectory is retained in the internal audit artifacts.

Algorithm 1 summarizes the automated update applied before cycle j (§2.3). Our experiments deploy K_j directly; in production, an optional human-review checkpoint may gate its deployment (§E).

B.2 Velocity Summary Computation

The velocity block is computed over the recent transaction history as follows. Timestamps are parsed to UTC datetimes; inter-arrival intervals are computed between consecutive transactions and summarised as minimum and median (in minutes). Counts in tail windows (last 10 and 30 minutes before the current transaction) are computed by scanning the sorted timestamp list. Unique-set diversities of multiple important fields are computed from the respective raw fields after stripping null/-NaN values. Historical amounts are extracted and summarized.

B.3 Decision Agent Prompt

Decision Agent System Prompt (abridged; platform and proprietary feature identifiers anonymized)

Role. You are a fraud-detection assistant at [Platform], reviewing a transaction. The user has m prior successful transactions, but recency does not guarantee trust. A compromised account or a card-testing burst can produce a deceptively consistent history. Decide ALLOW or BLOCK; "fraud" means the user is likely to dispute the transaction (financial loss).

Risk scale. 5 Critical (block); 4 High (block/verify); 3 Medium (add friction); 2 Low (allow+monitor); 1 Very Low (allow).

Data sources. transaction/checkout fields; [provider's signals]; [provider's signals] (feature names anonymized).

History. up to 15 most recent transactions (oldest→newest, each annotated with elapsed time) plus a computed burst/velocity block; treated as context, not automatic evidence of legitimacy.

Output. exactly one line: `RISK_LEVEL: <1-5>`, with no analysis or explanation.

The parser takes the first `RISK_LEVEL:\s*([1-5])` match; a response with no match is treated as *allow* (never a block), so a malformed generation cannot silently raise the block rate.

B.4 Reflection Agent Prompt

Reflection Agent Instructions (abridged schema; leaf-feature values anonymized)

Role. During each offline cycle, construct a structured hypothesis space indicating *where to look* for boundary errors. The agent may refine its proposal over multiple model calls, but only the deterministic verifier can admit or retire a rule.

Evidence digest (input). matured per-cycle operating statistics; the composition of the risk-3 bank; confirmed fraud that remains allowed under the current knowledge state; and fraud-vs-legitimate profiles for eligible features.

Final output (strict JSON). `{"amount_grid":[...], "qualifiers":[...], "release_grid":[...], "rationale":"..."}`. The `rationale` is retained for audit but never executed. `amount_grid`: permitted numeric atoms; `qualifiers`: up to ∼30 single leaf-feature atoms; `release_grid`: a few conjunctive release bands.

Atom grammar. Single atoms are numeric `FIELD OP VALUE` ($\ge, \le, >, <, =$), categorical equalities over the closed leaf-feature set, or named Boolean flags. Only `release_grid` may join permitted atoms with `&`; the harness constructs escalation conjunctions from `amount_grid` and `qualifiers`. Any atom outside the fixed vocabulary is dropped.

Constraints. Propose candidate feature families and ranges, not deployable decisions; prefer general leaf-level qualifiers; and do not bind proposals to the decision agent's emitted risk level.

C Experimental Setup and Baseline Details

C.1 Benchmark Construction

The production payment-risk stream is proprietary and cannot be released. We characterize the fixed evaluation cohort used for all reported results 6. It contains 16,140 upstream-allowed transactions with matured outcomes, including 813 confirmed fraud cases (5.04% of the benchmark). Company policy prevents disclosure of the broader-stream distribution and sampling statistics. Direct identifiers and label-leaking fields are removed before serialization, and point-in-time prior history is preserved for every retained transaction.

The cohort is a case–control sample: all matured fraud is retained and legitimate traffic is subsampled, so 5.04% is not the production prevalence. All systems score the same 16,140 transactions. Recall is invariant to this negative subsampling, whereas precision, F1, and block rate describe the benchmark operating point. Production prevalence, routing criteria, and traffic coverage remain proprietary. We therefore interpret results conditionally on this cohort rather than as end-to-end production precision or block rates.

For chronological replay the benchmark is ordered by timestamp and partitioned into nine weekly cycles (§3.2). Model training and operating-point selection for all systems use earlier, disjoint periods of the same stream that are not part of the evaluation cohort. All splits are disjoint at the transaction level, with no row-level overlap by transaction ID or by the pair of entity hash and timestamp; recurring entities may appear across periods, as expected in a streaming payment setting, and are used only to construct prior-history features from events preceding the current transaction.

Data availability. Legal and compliance restrictions prevent release of transaction-level records. To support scrutiny without exposing customer data, we report aggregate cohort statistics and provide the replay protocol, prompts, model settings, and verification procedure in the paper and appendix.

Prior history	Transactions	Fraud	Fraud rate
≥ 0 (all)	16,140	813	5.04%
≥ 1 prior	10,263	516	5.03%
≥ 2 prior	7,391	346	4.68%
≥ 3 prior	5,650	223	3.95%

Table 6: Evaluation-cohort composition by available prior-history depth.

C.2 Baseline Implementations

C.2.1 Supervised CatBoost

We train CatBoost on an earlier period using more than 100 features, including rolling user-history aggregates. Identifiers and post-outcome fields are excluded, while categorical variables are consumed natively. We use a positive- class weight and early stopping on the subsequent validation period.

We run 50 hyperparameter-optimization trials with Optuna. The search space covers the principal CatBoost capacity, step-size, sampling, and regularization parameters, including tree depth, learning rate, L2 leaf regularization, random strength, bagging temperature, and feature subsampling. We select the trial with the highest validation PR-AUC. For the operating-point comparison, we evaluate the continuous CatBoost score at a 5.0% block-rate operating point. Table 2 reports this operating point alongside the realized block rates of the discrete LLM actions.

To reflect production adaptation, **CatBoost-periodical** retrains the same model monthly over the evaluation stream. Retraining respects the same two-cycle maturation lag as SR-Fraud, and the validation split is used only for early stopping.

C.2.2 Supervised Sequence Model (Seq-GRU)

Seq-GRU is a matched-input supervised baseline that controls whether the gain comes from access to episodic history or from reasoning over it. It receives the current transaction and up to $k{=}10$ strictly earlier events from the same history index, using the CatBoost feature set with rolling aggregates disabled. A per-event ReLU–dropout projection feeds a three-layer GRU (hidden size 128, packed variable-length sequences) and binary classification head. We train with class-weighted binary cross-entropy and Adam, selecting hyperparameters and early-stopping on validation average precision. Seq-GRU emits a continuous score, so as with CatBoost we evaluate it at a 5.0% block-rate operating point on the evaluation cohort.

C.2.3 LLM Comparison Systems

The **Direct LLM** receives only the current transaction serialized as text (Hegselmann et al., 2023); it is the no-history control for the model's single-transaction prior. The **Decision Agent-only** uses the same online decision procedure but addition-

ally receives the Hybrid Episodic Window $W_u(t)$, enabling comparison against the entity's recent behavior. Its knowledge state is $K_\varnothing$, so it performs no outcome-supervised adaptation.

C.2.4 Reflexion

Fraud decisions cannot literally reproduce Reflexion's attempt–feedback–retry loop because each transaction is scored only once. We therefore map an episode to one week: the agent scores that week's transactions, waits for their outcomes to mature, writes a verbal reflection on its mistakes, and carries the resulting memory into subsequent weeks. This preserves the attempt–feedback–reflection structure across successive non-retriable batches.

The baseline uses the same weekly cadence and mechanically enforced maturation wall as SR-Fraud: decisions in cycle j use knowledge derived only from cycles matured by $j-2$. Its reflection receives the same evidence as SR-Fraud: confirmed false negatives, the observable risk-3 boundary traffic used for rejection inference, and the week's highest-dollar confirmed fraud. At decision time, the memory is static context in the system prompt, and each transaction requires one LLM call; memory writing occurs offline once per week and adds no iterative calls to the live path.

The deliberate contrast variable is *verification*. Reflexion maintains a single running free-text knowledge document, which the model rewrites weekly to merge new patterns and remove stale ones. Whatever the model concludes is placed directly into the following week's prompt. Unlike SR-Fraud, candidate rules receive no legitimate-traffic rarity screen, fraud-enrichment test, or promotion gate conditioned on matured precision. We use the least-guarded reflection prompt, without instructions that approximate these checks, so the comparison isolates unverified verbal reflection rather than a partially self-verified variant.

C.3 Model Configuration

We use `claude-opus-4-5` (Anthropic, 2025), `claude-sonnet-4-6` (Anthropic, 2026), and `gpt-5.4` (OpenAI, 2026) through an OpenAI-compatible gateway. Single-pass scoring uses no extended thinking, temperature 0.3 where supported, and default top-p. We extract the first `RISK_LEVEL: <1-5>` match; an unparseable response is treated as allow.

	Prec.	Rec.	F1	Recall$_\$$
≥ 0 *prior (all)*				
Δ vs Dec. agent	**+5.4** [2.8,8.1]	**+7.3** [4.0,10.7]	**+6.2** [3.6,9.0]	**+19.3** [14.3,24.5]
Δ vs CatB. period.	+0.3 [-4.7,5.1]	**+8.1** [1.7,14.4]	+3.8 [-1.5,8.8]	**+7.8** [0.6,15.2]
≥ 1 *prior*				
Δ vs Dec. agent	+2.0 [-1.7,5.4]	+3.5 [-0.2,7.4]	+2.7 [-0.6,5.9]	**+12.4** [7.0,18.3]
Δ vs CatB. period.	+3.7 [-3.0,10.3]	**+16.9** [9.0,24.7]	**+10.4** [3.3,17.2]	**+14.4** [5.1,24.1]
≥ 2 *prior*				
Δ vs Dec. agent	0.0 [-4.4,4.2]	+0.6 [-3.8,5.2]	+0.2 [-3.7,4.1]	**+7.9** [0.3,15.3]
Δ vs CatB. period.	+6.4 [-2.1,15.5]	**+24.0** [13.9,34.6]	**+15.7** [7.1,25.1]	**+23.6** [11.7,36.0]
≥ 3 *prior*				
Δ vs Dec. agent	+0.6 [-5.1,6.4]	+1.8 [-4.0,7.3]	+1.1 [-4.0,6.0]	+10.4 [-0.4,20.8]
Δ vs CatB. period.	+9.9 [-1.2,20.9]	**+26.5** [14.6,38.4]	**+18.9** [8.6,29.3]	**+29.9** [14.5,44.8]

Table 7: Paired SR-Fraud-minus-baseline differences with entity-clustered bootstrap 95% confidence intervals. Bold entries exclude zero.

D Additional Experiments

The supplementary experiments are organized into mechanism checks, verifier robustness analyses, and deployment-efficiency comparisons.

D.1 Statistical Reliability

We quantify the reliability of SR-Fraud's improvements with an entity-clustered bootstrap (B=2000). Entities are resampled with replacement, and the same resampled entities are used for both systems so that each paired difference accounts for the shared sample. Table 7 reports the SR-Fraud-minus-baseline differences and their 95% CIs by prior-history cohort; bold entries denote intervals that exclude zero.

The reflection gain over the decision agent is reliable on the full benchmark: all four differences are positive with CIs above zero, including $+6.2$ F1 $[3.6, 9.0]$ and $+19.3$ Recall$_\$$ $[14.3, 24.5]$. Within smaller history cohorts, the precision/recall/F1 intervals include zero. The dollar-recall gain excludes zero for ≥ 1 ($+12.4$ $[7.0, 18.3]$) and ≥ 2 ($+7.9$ $[0.3, 15.3]$), while the ≥ 3 interval includes zero ($+10.4$ $[-0.4, 20.8]$).

The comparison with CatBoost-periodical follows a different pattern. The methods have indistinguishable precision and F1 on the full benchmark, while SR-Fraud's recall and dollar recall are significantly higher. From the ≥ 1 cohort onward, recall,

F1, and dollar-recall intervals exclude zero and widen with history depth, reaching $+29.9$ Recall$_\$$ $[14.5, 44.8]$ at ≥ 3 prior transactions. Thus, reflection provides a robust overall gain over the frozen decision agent, while the LLM's advantage over the tabular model concentrates where behavioral history is deepest (§3.4.2).

D.2 History-Content Utilization

Section 3.4.2 shows that history *presence* helps. We further test whether the decision agent uses the *content* of the Hybrid Episodic Window, or whether any plausible-looking history would do. On the $\geq$3-prior-history cohort (5,650 transactions, 223 fraud, 1,771 entities), we score the identical frozen agent (Sonnet 4.6, block at risk ≥ 4) on the identical current transactions under three history conditions: **correct** (the transaction's real window), **shuffled** (the recent history of a different, randomly chosen entity, with the current-transaction fields left unchanged; verified 0 self-donors), and **no-history** (current transaction only). Only the history block changes.

History	ΔP	ΔR	ΔF1	ΔR$_\$$	Block
CatB.-period. (ref.)	0.0	0.0	0.0	0.0	N/A
Correct	+9.3	+24.7	+17.8	+19.5	4.50%
Shuffled	-24.9	+53.8	-14.7	+51.6	60.6%
No-history	-9.1	-9.8	-10.3	-11.8	1.89%

Table 8: History-content ablation as percentage-point differences from CatBoost-periodical on the ≥ 3 history cohort.

The history-content result is also visible relative to CatBoost-periodical. Correct history improves precision, recall, F1, and dollar-weighted recall by $+9.3$, $+24.7$, $+17.8$, and $+19.5$ points. Wrong-entity history produces a large negative precision delta and a 60.6% block rate, whereas removing history yields negative recall and dollar-recall deltas. The same pattern holds across three donor seeds and with amount-matched donors, confirming that the agent uses entity-specific behavior rather than a generic history indicator.

D.3 Proposer Ablation

We ask whether a classical rule-discovery algorithm can replace the LLM in the weekly proposal stage. All runs use the same decision agent (Sonnet 4.6), maturation wall, verifier, admission budget, and rejection-inference release. Each rule miner independently constructs its pre-replay state and weekly updates; none is initialized from LLM-authored rules or state.

Brute-force search is infeasible on raw features. Direct search over the raw feature space faces two practical obstacles (Table 9). First, discretization produces 527 atoms, making an arity-3 enumeration 2.4×10^7 candidates per cycle, nearly $3{,}000\times$ the 8,473 candidates in the distilled grammar. Every candidate would also need weekly re-verification on matured evidence. Second, a tractable beam approximation over-specializes: greedily extending the highest-precision conjunctions produces extremely narrow rules. The beam policy attains high marginal precision on the transactions it blocks, but covers under 0.1% of benchmark traffic and correspondingly loses 31.0 F1 points and 48.2 dollar-recall points relative to static CatBoost. Greedy precision maximization therefore converges on rules that are pure but operationally inert. The resulting policy fires on just 15 transactions, yielding high purity but near-zero coverage.

Comparison with established rule miners. We next evaluate three established non-LLM miners as drop-in proposers: **FP-Growth** (Han et al., 2000), which mines frequent itemsets among fraud examples and ranks them by lift; **RIPPER** (Cohen, 1995), an imbalance-aware sequential-covering method; and **WRAcc subgroup discovery** (Han et al., 2000), which performs beam search using weighted relative accuracy. Each method constructs its own initial state from the training and validation splits and proposes weekly updates through the shared harness. We evaluate each miner on both the reflection agent's selected feature subset and the full raw feature set. Two simple controls provide lower bounds: a random-vocabulary initialization built on the training and validation splits and then held fixed during replay, and CART induction, which finds no fraud-enriched leaf under the 0.6% training prevalence even at depth 8. Table 10 reports the resulting policies.

Two patterns emerge. First, SR-Fraud yields the strongest F1 and dollar-recall point estimates among the evaluated proposers. It reaches an F1 delta of $+8.0$ and a dollar-recall delta of $+11.7$ points, versus at most $+5.4$ and $+9.6$ points, respectively, across the six classical-miner configurations. The simple controls provide little additional evidence: the frozen random initialization changes F1 by only 0.2 points relative to no reflection, and

	Distilled grammar	Raw features
Atom pool size	37	527
Arity-3 candidates / cycle	8,473	2.4×10^7

Table 9: The raw feature space a deterministic proposer faces without the LLM. Exhaustive enumeration over the raw atoms is intractable (2.4×10^7 candidates/cycle).

Proposer (full loop, Sonnet dec.)	**ΔP**	**ΔR**	**ΔF1**	**ΔR$_\$$**	**Block**
CatBoost (reference)	0.0	0.0	0.0	0.0	5.00%
No reflection ($K_\varnothing$)	-0.9	+5.0	+1.8	-7.6	5.97%
Random vocabulary (frozen)	-2.3	+7.9	+2.0	-1.5	6.75%
Decision-tree induction	*no fraud-enriched leaf found*				n/a
Rule miners over the reflection agent's features					
FP-Growth	+1.9	+8.2	+4.8	+4.3	5.94%
RIPPER	+1.1	+8.1	+4.3	+2.1	6.06%
WRAcc subgroup	+0.8	+11.4	+5.4	+9.6	6.64%
Rule miners over raw features					
FP-Growth	+1.9	+5.4	+3.6	-2.3	5.52%
RIPPER	+0.8	+7.2	+3.8	+0.9	5.99%
WRAcc subgroup	+4.8	+0.9	+2.7	-11.1	4.47%
SR-Fraud (LLM proposer)	+4.5	+12.3	+8.0	+11.7	6.08%

Table 10: Proposer comparison as percentage-point differences from static CatBoost. Each rule miner constructs its own initial state and weekly proposals; the decision agent, maturation wall, verifier, admission budget, and release mechanism are shared. The independently built random-vocabulary control is held fixed during replay. Each miner is evaluated on the agent-selected and raw feature sets. Operating points are not friction-matched.

CART finds no usable leaf.

Second, the reflection agent's selected feature subset improves both F1 and dollar recall for every classical miner. For FP-Growth, RIPPER, and WRAcc, respectively, the selected/raw F1 deltas are $+4.8/+3.6$, $+4.3/+3.8$, and $+5.4/+2.7$ points; the corresponding dollar-recall deltas are $+4.3/-2.3$, $+2.1/+0.9$, and $+9.6/-11.1$ points. The raw-feature WRAcc run is especially weak on dollar recall because its broad subgroups fail the shared evidence gates. Thus, adding raw features does not close the gap. Together, these results support two complementary roles for the LLM: discovering a compact feature subset and proposing transferable rules within it.

D.4 Reflexion Operating-Point Sensitivity

Because the decision agent emits a discrete risk level in $\{1, \ldots, 5\}$ rather than a continuous score, the Reflexion baseline cannot be thresholded to an exact 5% block budget; only two natural cut points exist: risk ≥ 4 (SR-Fraud's block rule) and the most conservative risk ≥ 5. Table 11 reports both, for all three backbones, under the identical weekly loop and maturation wall of §C.2.4 (each backbone serves as both the decision agent and the reflection author).

At risk ≥ 4, every Reflexion backbone exceeds the target block budget, with realized block rates ranging from 14.8% to 41.4%. Raising the threshold to risk ≥ 5 does not provide a consistent remedy: Sonnet moves near budget, Opus becomes overly conservative, and GPT-5.4 still blocks well above budget. The free-text memory therefore inherits each backbone's raw escalation tendency. SR-Fraud instead bounds policy changes through matured-evidence admission and a fixed per-cycle cap. Full SR-Fraud is evaluated with Sonnet 4.6 only, so the cross-backbone comparison is used to diagnose Reflexion rather than to claim measured SR-Fraud performance on every backbone.

D.5 CatBoost at SR-Fraud's Block Budget

The ordinal LLM policy does not provide a principled within-tier ranking for forcing it to an arbitrary 5% budget. We therefore leave SR-Fraud completely unchanged and instead evaluate the continuous CatBoost scores at SR-Fraud's native operating point 12. Full SR-Fraud blocks 982 of $16{,}140$ transactions (6.0843%, reported as 6.08%). For each CatBoost variant, we rank transactions by its continuous score and block exactly the top 982;

Backbone	Cut	ΔPrec.	ΔRec.	ΔF1	$\Delta\text{Recall}_\$$	Block
CatBoost (reference)	5%	0.0	0.0	0.0	0.0	5.00%
Sonnet 4.6	≥ 4	-13.3	+23.5	-3.9	+26.1	14.80%
Sonnet 4.6	≥ 5	+7.0	+1.8	+4.2	-6.6	4.34%
Opus 4.5	≥ 4	-22.2	+31.7	-14.8	+19.5	32.24%
Opus 4.5	≥ 5	-10.7	-15.8	-13.6	-30.0	3.79%
GPT-5.4	≥ 4	-25.1	+25.4	-19.6	+10.8	41.44%
GPT-5.4	≥ 5	-22.7	-9.5	-18.8	-21.3	12.06%
SR-Fraud (Sonnet 4.6)	≥ 4	**+4.5**	**+12.3**	**+8.0**	**+11.7**	6.08%

Table 11: Reflexion operating points as percentage-point differences from static CatBoost.

labels are used only after selection to compute metrics. No SR-Fraud decision or evaluation row is removed.

Method	ΔP	ΔR	ΔF1	$\Delta\text{R}_\$$	Block
CatBoost (reference)	0.0	0.0	0.0	0.0	6.08%
CatBoost-periodical	+3.8	+4.6	+4.1	+5.4	6.08%
SR-Fraud	**+6.8**	**+8.2**	**+7.5**	**+7.3**	6.08%

Table 12: Performance at a common block count as percentage-point differences from the budget-matched CatBoost reference.

Method	Week-8 burst		High-value vector	
	ΔR	$\Delta\text{R}_\$$	ΔR	$\Delta\text{R}_\$$
CatBoost (reference)	0.0	0.0	0.0	0.0
CatBoost-periodical	+10.2	+6.3	-1.8	-0.2
SR-Fraud	**+34.5**	**+35.3**	**+9.1**	**+8.7**

Table 13: Subset performance at the common 6.08% block count as percentage-point differences from CatBoost.

Table 13 shows the same point-estimate ordering on the burst and high-value subsets under the exact matched budget.

At the common block count, SR-Fraud has higher point estimates than both tree baselines on all four metrics. An entity-clustered paired bootstrap ($B = 2000$) gives SR-Fraud-minus-static-CatBoost differences of $+6.8$ precision $[2.1, 11.5]$, $+8.2$ recall $[1.3, 14.6]$, $+7.5$ F1 $[2.1, 12.7]$, and $+7.3$ dollar recall $[-1.1, 15.2]$. Against CatBoost-periodical, the differences are $+3.1$ precision $[-1.8, 7.6]$, $+3.7$ recall $[-2.8, 10.4]$, $+3.3$ F1 $[-1.8, 8.4]$, and $+1.9$ dollar recall $[-5.4, 9.1]$. Thus, after exact block-count matching, SR-Fraud's precision, recall, and F1 advantages over static CatBoost are significant (dollar recall is positive but its interval includes zero), while none of its differences from CatBoost-periodical is statistically significant.

D.6 Reflection- and Decision-Backbone Combinations

The main results pair each decision agent with its own model as the reflection agent (the diagonal, $R = D$). Here we vary the two roles independently: the reflection agent R (which proposes boundary hypotheses each cycle) and the decision agent D (which produces the frozen risk scores). All cells use the same deterministic verifier, rejection-inference release, and per-cycle admission cap.

Reflect. R	Decision D	ΔP	ΔR	ΔF1	$\Delta\text{R}_\$$
Sonnet 4.6	Sonnet 4.6	+4.5	+12.3	+8.0	+11.7
	Opus 4.5	+6.4	+6.0	+6.2	+1.2
	GPT-5.4	-1.0	+6.3	+2.3	+1.4
Opus 4.5	Sonnet 4.6	+5.3	+12.7	+8.7	+13.2
	Opus 4.5	+6.2	+3.0	+4.6	-2.2
	GPT-5.4	-1.3	+3.4	+0.9	-3.8
GPT-5.4	Sonnet 4.6	+4.9	+11.8	+8.1	+8.7
	Opus 4.5	+5.9	+5.2	+5.5	+0.8
	GPT-5.4	-1.2	+7.6	+2.7	+2.8

Table 14: Reflection (R) $\times$ decision (D) backbone matrix as percentage-point differences from static CatBoost. Each cell runs the same propose–verify loop and rejection-inference release. The main table reports the diagonal ($R{=}D$).

Two patterns emerge. The *decision* agent sets the operating point (columns): $D =$ Sonnet yields the strongest F1/dollar-recall for every reflector, because it routes the most fraud into the reachable $r{=}3$ band, whereas $D =$ Opus and $D =$ GPT leave a larger share at $r \leq 2$ (out of reflection's reach) and are correspondingly capped. The *reflection* agent matters much less within a column: with $D =$ Sonnet, the F1 deltas are $+8.0/+8.7/+8.1$ points for $R \in \{\text{Sonnet, Opus, GPT}\}$, indicating that the propose–verify–refine loop is robust to the proposer backbone.

D.7 Rejection-Inference Validation

Across all replay cycles, the rejection-inference proxy and oracle labels select the same anonymized release family, with the proxy choosing a slightly broader band than the oracle. Relative to static CatBoost, the proxy configuration has recall and dollar-recall deltas of $+12.3$ and $+11.7$ points; with oracle labels, they become $+15.0$ and $+14.2$ points. The block rate changes from 6.08% to 6.55%, and the selected release family never changes. Because release is restricted to the matching risk-4 slice, the resulting policy difference remains localized. This agreement supports the risk-3 bank as a practical proxy, while not implying that it identifies the unobserved counterfactual labels of individual blocked transactions.

D.8 Verification Confidence Sensitivity

The admission and release bounds use a one-sided Wilson $z = 1.0$. We re-verify the persisted main-run candidates under $z \in \{1.64, 1.96, 2.58\}$ while holding the remaining pipeline fixed; z=1.0 is the reported run. The additional values are common standard-normal critical values spanning approximately 95% to 99.5% one-sided coverage; they serve as sensitivity settings rather than family-wise confidence guarantees. Table 15 shows that the operating point remains stable across these settings.

Setting	Final rules	ΔP	ΔR	ΔF1	$\Delta R_{\$}$	Block
CatB. (ref.)	–	0.0	0.0	0.0	0.0	5.00%
$z = 1.0$	11	+4.5	+12.3	+8.0	+11.7	6.08%
$z = 1.64$	6	+3.8	+10.9	+7.1	+8.0	6.00%
$z = 1.96$	6	+3.8	+10.9	+7.1	+8.0	6.00%
$z = 2.58$	6	+4.2	+9.7	+6.8	+6.6	5.76%

Table 15: Wilson-bound sensitivity as percentage-point differences from static CatBoost. Candidates are re-verified with only z changed.

D.9 Candidate Admissions and Multiplicity

Table 16 reports the proposal–verification funnel for the headline Sonnet configuration. The persisted atom counts are the unions across iterative proposal rounds. Candidate checks count verifier evaluations across rounds and therefore need not be unique rules. A verified candidate clears the evidence gate; cap-excluded candidates clear that gate but fall outside the fixed per-cycle admission budget.

Table 17 traces the active-signature count per cycle for each self-reflection configuration ($R = D$), together with the total admissions and retirements. Because r=3 evidence only matures after the two-week wall, no signature can be corroborated before cycle 4; Opus and GPT first grow at cycle 4, whereas Sonnet first admits new signatures at cycle 5. Each admits its verified families over the following cycles. The three backbones differ sharply in admission behavior: GPT admits the maximum 5 signatures at every growth cycle (20 total, cap-bound), Opus admits 11 without ever retiring, and Sonnet is the most conservative (8 admissions) and the only configuration to retire (-2 at cycle 6, so its count dips from 10 to 8 before regrowing to 11). Per-cycle corroboration and the fixed cap limit growth but are not a formal multiple-testing correction. The *aggregate* effect of these online admissions is quantified in Table 18: relative to the evolved system, holding the pre-replay state fixed reduces F1 by 0.4/2.4/1.7 points and dollar-recall by 4.3/11.2/5.9 points for Sonnet/Opus/GPT.

D.10 Online Reflection vs. a Static State

To isolate the contribution of *online* reflection, we compare each evolved configuration against a control that holds fixed the pre-replay knowledge state constructed from the training and validation splits. The decision agent, initialization procedure, verifier, and rejection-inference release are otherwise unchanged. Table 18 reports both. Online reflection improves dollar-recall for every backbone, especially where the static state is weakest relative to the decision agent's reachable pool, confirming that the per-cycle propose–verify updates—rather than a one-time offline construction—drive the gains.

D.11 Request-Time Latency Analysis

SR-Fraud keeps reflection off the request path. Each decision uses one stateless LLM call, after which the knowledge state K is applied deterministically. Table 19 compares this single-pass design with Reflexion and an inference-time reasoning agent, using Sonnet 4.6 and the same Hybrid Episodic Window. Single-pass SR-Fraud and Reflexion remain near 1.3 s at p_{50}, whereas chain-of-thought raises latency to 47.3 s and greatly increases output length. SR-Fraud instead amortizes reasoning into the offline reflection cycle, improving the policy without adding request-time reasoning calls; its distinction from Reflexion is verification quality rather than latency.

Cycle	Atoms A/Q/R	Checks	Verified	Cap-excl.	Admitted	Retired
3	8/33/3	1,092	0	0	0	0
4	10/34/5	1,449	0	0	0	0
5	10/34/5	1,612	7	2	5	0
6	10/36/5	1,378	0	0	0	2
7	11/36/5	1,467	1	0	1	0
8	11/37/4	1,440	0	0	0	0
9	11/37/5	1,351	2	0	2	0
Total	–	**9,789**	**10**	**2**	**8**	**2**

Table 16: Headline proposal–verification funnel by replay cycle. A/Q/R denotes amount atoms, qualifier atoms, and release bands. Checks are verifier evaluations across iterative rounds, not deduplicated candidate rules.

Active sigs. after cycle	1	3	4	5	6	7	9	Σadm.	Σret.
Sonnet×Sonnet	5	5	5	10	8	9	11	8	2
Opus×Opus	9	9	14	19	19	19	20	11	0
GPT×GPT	5	5	10	10	15	20	25	20	0

Table 17: Active escalation-signature count per reflection cycle for each self-reflection ($R = D$) configuration, from the state deployed at the start of replay through the state after the final cycle, with total admissions (Σadm.) and retirements (Σret.). Growth is gated by per-cycle corroboration on matured evidence and a fixed admission cap of 5 per cycle. GPT saturates that cap at every growth cycle ($+5$ each), indicating it proposes more corroborated candidates than the cap admits; Sonnet is the most selective and the only configuration to retire rules (-2 at cycle 6 on later evidence).

Config ($R = D$)	**State**	**ΔP**	**ΔR**	**ΔF1**	**ΔR$_\$$**
Sonnet 4.6	Static state (reference)	0.0	0.0	0.0	0.0
	Evolved state (SR-Fraud)	+0.4	+0.5	+0.4	+4.3
Opus 4.5	Static state (reference)	0.0	0.0	0.0	0.0
	Evolved state (SR-Fraud)	0.0	+4.2	+2.4	+11.2
GPT-5.4	Static state (reference)	0.0	0.0	0.0	0.0
	Evolved state (SR-Fraud)	+1.4	+2.1	+1.7	+5.9

Table 18: Online reflection relative to a static-state control for each self-reflection configuration. Values are within-backbone percentage-point differences; the decision agent, initialization procedure, verifier, and release mechanism are matched.

Configuration	p_{50} **(s)**	p_{95} **(s)**	**Out tok**
Single-pass / SR-Fraud	1.31	2.41	11
Reflexion (prompt memory)	1.38	3.91	11
Reasoning/CoT agent	47.3	61.5	2,484

Table 19: Request-time latency and output length for single-pass, prompt-memory, and chain-of-thought decision agents.

D.12 Post-Training and Deployment Analysis

Newer closed models do not guarantee better fraud detection. Across three model families, the newer closed release fails to improve fraud detection (Table 20). Opus 5 and Sonnet 5 regress on all four metrics versus Opus 4.5 and Sonnet 4.6; the newer GPT-5.6 (sol) variant raises precision only by becoming far more conservative, leaving its F1 and dollar-recall well below GPT-5.4. In every family the newer model lowers recall, F1, and dollar-recall. This non-monotonic behavior motivates adapting a self-hosted model whose training and deployment are under system control.

Decision-agent	**ΔP**	**ΔR**	**ΔF1**	**ΔR$_\$$**
CatBoost (reference)	0.0	0.0	0.0	0.0
Opus 4.5	-2.1	-1.5	-1.8	-15.3
Opus 5	-10.8	-2.3	-7.3	-17.2
Sonnet 4.6	-0.9	+5.0	+1.8	-7.6
Sonnet 5	-11.9	-17.2	-15.0	-30.5
GPT-5.4	-11.8	+2.8	-6.4	-11.0
GPT-5.6 (sol)	-0.3	-17.9	-12.5	-32.8

Table 20: Successive closed-model generations as percentage-point differences from static CatBoost.

Training data and reward. We post-train `Qwen/Qwen3.5-9B` (Qwen Team, 2026) on prompts rendered in the same transaction format as evaluation. The training split retains all fraud cases and samples three non-fraud transactions per fraud case, yielding roughly 4,000 prompts. The reported **SR-Fraud-PT** row is this post-trained model evaluated as a single-pass decision agent at the same $r \geq 4$ operating point, with no symbolic knowledge state applied; it thus isolates weight-encoded adaptation and is complementary to, not combined with, the reflection layer. Post-training is essential: the untrained base 9B scores far below every other system (Table 5), so the gain reflects the domain adaptation rather than the backbone's zero-shot ability. The reward maps risk levels 1–5 to signed fraud

confidence $(-1, -0.5, 0, 0.5, 1)$ and applies class weights 3 for fraud and 1 for non-fraud, so constant policies have zero expected class-balanced reward.

Optimization. We trained Qwen3.5-9B using GRPO (Shao et al., 2024) with eight completions per prompt and LoRA (Hu et al., 2022) on all linear layers (rank 32, $\alpha = 64$, Kaiming initialization). The run uses three epochs, train batch size 64, learning rate 5×10^{-5}, and KL coefficient 10^{-3}. It is then evaluated with vLLM 0.20.2 in bfloat16.

Detection and adaptability. Post-training allows the smaller self-hosted model to reach a detection profile similar to the commercial SR-Fraud configuration and supports local GPU serving. Its knowledge, however, is encoded in model weights: tracking new fraud patterns requires another training and deployment cycle. In contrast, SR-Fraud can incorporate verified symbolic rules between model refreshes. The two mechanisms are therefore complementary: post-training provides a strong base policy, while reflection supplies an auditable adaptation channel between retraining cycles.

Serving and latency measurement. We serve the post-trained 9B decision agent locally with vLLM 0.20.2 in bfloat16 on a single NVIDIA L40S GPU, exposing an OpenAI-compatible endpoint. We measure request-time latency on 200 held-out prompts at concurrency 10, timing each call from submission to the first-line risk verdict. On this setup the self-hosted agent has a median request latency of 0.74 s and a p_{90} of 0.94 s (Table 5), below the corresponding p_{90} of the hosted Sonnet 4.6 decision agent and full SR-Fraud (2.02 s) reported in Table 2, which are billed per token and served through a hosted API. A self-hosted model therefore trades per-token cost for provisioned GPU capacity while keeping the single request-time LLM call well within the authorization-path latency budget.

Cost trade-off. The two serving modes have opposite cost structures: a hosted API charges a flat per-token amount per call, whereas a self-hosted GPU is a fixed daily cost amortized over the day's calls. At 6,000 input and 11 output tokens per call, a call costs 1.82 cents on Sonnet ($3/$15 per 1M in/out tokens) and 3.03 cents on Opus ($5/$25). Serving the 9B agent on one GPU costs $34/day (L40S reserved), $54/day (L40S on-demand), or $81/day (RTX Pro 6000 on-demand), amortized over the day's volume. The measured throughput of 12.96 req/s at concurrency 10 corresponds to approximately 1.1M calls/day under continuous utilization, so all volumes considered here fit on one GPU. Table 21 reports the resulting per-call cost as a fraction of the Sonnet and Opus API prices: reserved self-hosting already undercuts Sonnet at 2,500 calls/day, and by 10,000 calls/day every option costs well under a third of the Sonnet price and under a fifth of Opus. Beyond cost, self-hosting avoids sending transaction prompts to an external model API and reduces provider-drift risk.

Self-hosted 9B	**Calls / day**			
(× Sonnet / Opus)	2,500	5,000	7,500	10,000
L40S (reserved)	0.75 / 0.45	0.37 / 0.22	0.25 / 0.15	0.19 / 0.11
L40S (on-demand)	1.19 / 0.71	0.59 / 0.36	0.40 / 0.24	0.30 / 0.18
Pro 6000 (on-demand)	1.78 / 1.07	0.89 / 0.53	0.59 / 0.36	0.44 / 0.27

Table 21: Self-hosted 9B per-call cost relative to the Sonnet and Opus API price (6,000 input / 11 output tokens; Sonnet 1.82 cents, Opus 3.03 cents per call), as a multiple of each API's cost. Self-hosted cost is the fixed GPU $/day divided by daily volume on a single node; values below 1× mean self-hosting is cheaper.

E Additional Discussion

Why additive rather than end-to-end LLM? Two considerations argued against replacing the tabular classifier. *Latency*: the upstream classifier responds in tens of milliseconds; an LLM round-trip is slower. *Cost*: at sufficiently high traffic volumes, per-request LLM calls can dominate compute spend.

Why retain human review? The reported replay is fully automated. In production, an optional human checkpoint can review each knowledge-state changelog before deployment. Because updates are weekly and review occurs offline, this control does not affect request-time latency.