

Context-aware micro-escrow for agentic systems with language model validation

Ayman Nait Cherif¹, Mohamed Youssfi¹, Omar Bouattane²

¹2IACS Laboratory, ENSET Mohammedia, Hassan II University of Casablanca, Casablanca, Morocco

²IESI Laboratory, ENSET Mohammedia, Hassan II University of Casablanca, Casablanca, Morocco

Article Info

Article history:

Received Oct 28, 2025

Revised Jul 29, 2026

Accepted Aug 12, 2026

Keywords:

Agentic artificial intelligence

Agent-to-agent

Blockchain micro-escrow

Large language models

Micropayment

Model context protocol

Smart contract

ABSTRACT

Autonomous agents transacting with each other need an escrow they can trust without a human in the loop. This study propose a context-aware micro-escrow framework for this setting: large language models (LLMs), the model context protocol (MCP), agent-to-agent (A2A) communication, and blockchain smart contracts together support automated, auditable micro-transactions among heterogeneous resource holders such as live application programming interfaces (APIs), internet of things (IoT) sensors, and service providers. Traditional escrow relies on rigid condition checks or a trusted intermediary. This study instead runs a deterministic, checklist-based semantic validation pipeline that emits signed binary verdicts and commits rationale hashes on-chain, so any verdict can be replayed off-chain while on-chain settlement stays minimal. Context travels in cryptographically stamped MCP envelopes; negotiation and evidence streaming run over authenticated A2A channels. An auditing layer computes risk signals from recent settlement metadata and applies a bounded fee surcharge that internalizes monitoring effort and performance drift. The framework is evaluated with a reproducible simulator under benign and adversarial mixes, reporting latency, decision quality (precision and recall under a binary-classifier view), and economic side effects (fee collection and slashing). The results show strong tolerance to malicious providers. Coordinated validator compromise, however, marks a clear trust boundary, motivating validator-side hardening as future work.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Ayman Nait Cherif

2IACS Laboratory, ENSET Mohammedia, Hassan II University of Casablanca

Casablanca, Morocco

Email: ayman.nait.cherif@gmail.com

1. INTRODUCTION

Autonomous software agents have been studied for decades in distributed artificial intelligence (AI), where coordination was formalized in multi-agent systems research [1] through standards such as knowledge query and manipulation language (KQML) [2] and the Foundation for Intelligent Physical Agents Agent Communication Language (FIPA ACL) performatives [3]. While these standards were foundational, early multi-agent system implementations relied on broker-based middleware and lacked cryptographic guarantees of message provenance or contextual integrity. This limited their applicability in open, adversarial environments and motivated the shift toward more secure and verifiable coordination mechanisms.

Recent breakthroughs in AI have made large language models (LLMs) central to modern intelligent agents. LLMs exhibit human-level language understanding and reasoning in many domains. This lets agents reason over unstructured task descriptions and contextual constraints that cannot be captured by rigid

rule-based systems. For example, OpenAI's GPT-3 achieved near human performance on a range of benchmarks [4], inspiring a wave of research into “agentic” AI systems that use LLMs as autonomous decision-makers [5]. Unlike earlier agents bound by fixed rules or narrow scripts, modern LLM-based agents can dynamically plan actions, make decisions, and invoke tools in complex, open-ended environments [6]. This new paradigm of agentic AI endows agents with greater autonomy and the ability to collaborate with other agents toward a goal. In parallel, new protocols are emerging to support such multi-agent ecosystems, including context-sharing frameworks and agent-to-agent (A2A) communication standards that aim toward cross-platform interoperability. With this progress comes increasing recognition of the need for secure coordination: techniques such as zero-knowledge proofs combined with OAuth 2.0 [7] have been explored to attest facts without revealing sensitive details, and privacy-preserving protocols address agent anonymity and secure messaging [8]. These approaches extend trust guarantees from basic message exchange to the semantic level of agents' commitments.

What remains missing is infrastructure: autonomous agents still have no secure, scalable, and verifiable framework for economic micro-transactions. Yet the envisioned applications, spanning internet of things (IoT), finance, and beyond, assume countless small exchanges with minimal human oversight. An IoT device paying on demand for data, bandwidth, or energy is the canonical case [9]–[11]. Existing blockchain solutions hit well-known throughput and fee limits when micro-payments become frequent, and payment channel networks [12]–[15] move the load off-chain only by demanding pre-established channels, an awkward burden for resource-constrained nodes. Industry efforts such as the lightning service authentication token (LSAT)/L402 approach are only beginning to address the gap. A universally adopted micro-transaction framework for autonomous agents does not yet exist.

This paper address these challenges by proposing a context-aware micro-escrow framework that integrates LLM-based semantic validators, the model context protocol (MCP) for context data exchange, A2A secure communication, and blockchain smart contracts. When two agents negotiate an exchange, they route the transaction through a micro-escrow smart contract that holds payment until an LLM validator determines whether context-dependent conditions have been satisfied, with the result enforced on-chain. By combining LLM-driven semantic analysis with on-chain enforcement, the process becomes accountable and automation-friendly, reducing reliance on human arbitration while making validator behavior auditable through replayable evidence and signed verdict commitments. Every step, from the agents' initial agreement to the final payment outcome, is recorded either on-chain or in an append-only log, creating a tamper-evident audit trail compatible with privacy-preserving techniques such as pseudonyms, hashed content, and zero-knowledge proofs.

2. OVERALL ARCHITECTURE

2.1. Layered architectural overview

Five layers make up the architecture as shown in Figure 1. These layers are resource, agent, transport, trust-and-incentive, and governance-and-audit. The framework is organized this way to separate concerns and to keep it scalable; layers interact only vertically, through narrowly defined interfaces between adjacent layers, which is what enforces modularity and draws the security boundaries.

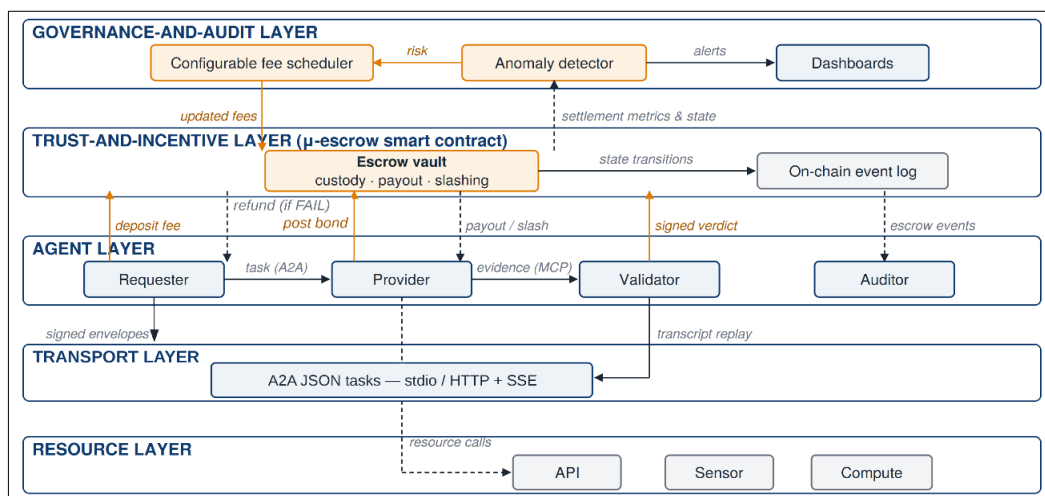


Figure 1. Layered architectural stack

A descriptor carries three things: input/output schemas, the required OAuth scope, and a recommended grounding prompt. Heterogeneous agents can therefore verify a capability offline and invoke tools consistently. Ambiguity about what a provider actually offers shrinks, and silent schema drift during execution, a common source of semantic disputes in automated service exchange, is blocked at the interface. Descriptors are versioned, so provider capabilities can grow without breaking existing consumers.

Agents live in the second layer, each with a pluggable LLM core and its own tool registry. Four roles structure the ecosystem: requester, provider, validator, and auditor. The split is deliberate. Responsibilities stay explicit, adversarial surfaces stay isolated from one another, and each role scales on its own: validator capacity can grow under load without touching provider logic or settlement rules.

Two wire formats are normative in the transport layer. Co-resident processes talk over a zero-latency studio channel; networked peers use HTTP POST with server-sent events for streaming. Why both? Local execution stays high-throughput, while cross-platform agent markets keep their wide-area interoperability. Discovery artifacts such as agent cards, capability ads, and intent offers travel only over authenticated channels, which is what blocks spoofing and capability poisoning.

Economic integrity sits in the trust-and-incentive layer. Its core is a deliberately minimalist smart-contract micro-escrow [16]: it holds deposits and provider bonds, exposes one oracle slot for a validator's signed verdict [17], and allocates micro-levies to the validator and auditor roles (the contract design and state machine are detailed in section 2.3). Funds come first. Because escrow precedes work, neither party can unilaterally reach the money before the agreed conditions are verified. The division of labor is strict, settlement logic on-chain and semantic interpretation off-chain, which buys trustless enforcement without ever asking the blockchain to process raw evidence.

Verification and risk control converge in the governance-and-audit layer. A deterministic validator replays the studio transcript or the server-sent events log, emits a binary verdict, and hashes its rationale, so audits work without raw content ever touching the chain. Two watchdogs run alongside it. A statistical anomaly detector reads verdict hashes and settlement metadata to flag drift and potential collusion; a dynamic fee engine moves premiums with observed risk.

Strictness at the interfaces is what the whole design leans on. Keep the interface contracts consistent, and any transport or blockchain can be swapped without touching agent logic. Cryptographic domains are distinct as well, isolating resource access from settlement logic so that a compromise has a small blast radius. The same property carries extensions: federated committees or proof-based verification can be introduced later without changing the settlement semantics.

2.2. End-to-end data and value flow

Seven phases carry a micro-escrow transaction from first contact to final payment: discovery, intent offer, escrow funding, secure context exchange, execution with streaming evidence, LLM-based validation, and settlement, as illustrated in Figure 2. Four roles do the work. Requester, provider, validator, and auditor are each scoped to their own subset of tasks; we drew the boundaries this way to keep formal analysis simple, to isolate the trust boundary at the validator, and to leave no cross-role attack surface. The payoff of the decomposition is traceability: every transition in the protocol maps to a specific agent action or on-chain event, which is what deterministic replay and audit rest on.

Discovery starts at the directories. The requester agent queries them with MCP descriptors and picks a provider by minimizing expected latency plus price; an exploration-exploitation strategy can optionally drive the choice. The offer itself, task and payment terms, is signed and sent over an A2A authenticated channel. On the other side, the provider agent has published signed MCP descriptors at a well-known endpoint advertising its capabilities, pricing, and required authorization scope; it accepts the offer and, where the protocol demands it, posts a performance bond [18] to the escrow contract. Then both parties jointly fund an on-chain smart-contract escrow instance, and payment stays locked until the settlement conditions are met.

Once funds are locked, context flows. The provider, authorized through a capability token or an OAuth scope, streams incremental results to the requester over the MCP channel, and every envelope is signed and hashed for integrity and provenance. Replay is handled at the door: incoming nonces and expiry timestamps are checked, and stale or replayed requests are rejected. Both agents log every exchanged envelope in an immutable, sequence-numbered record; that log becomes the tamper-evident evidence.

Delivery complete, the validator takes over as the semantic oracle. It replays the full interaction transcript deterministically and evaluates it against a declarative checklist, using a fixed prompt template at temperature 0. What leaves the validator is small: a signed binary verdict, pass or fail, plus a SHA-256 hash of the rationale — full auditability with no verbose text on-chain. Committee mode is optional and exists to tolerate faulty validators. An odd-numbered ensemble aggregates verdicts by weighted majority vote based on stake or reputation [19]; ties break toward rejection, and validators whose verdicts diverge from consensus are penalized [20].

Settlement is mechanical. On a pass, the contract releases payment to the provider and returns the bond; on a failure, it refunds the requester and slashes the bond into a treasury pool. Escrow ID, outcome, and evidence hash references land in an on-chain event log. Above individual escrows sits the auditor, watching continuously: it monitors completed transactions and computes per-provider anomaly scores.

Six stages formalize each role's progression as a finite state machine (FSM): idle, negotiating, executing, evidence-submitted, awaiting-verdict, and terminated. Nothing moves between stages without a verifiable event (escrow creation, envelope hash receipt, and verdict emission) arriving on-chain or over MCP, so ambiguous timeouts disappear and protocol progress becomes fully measurable. Agents that want to adapt strategy over time can plug in reinforcement learning modules restricted to public or hashed features. From this decomposition come the three guarantees the design aims at: i) verifiability: every decision is a deterministic function of signed or hashed data; ii) horizontal replicability: each role scales independently, with no bottleneck between them; and iii) incentive compatibility: bonds, deposits, and micro-fees make Byzantine deviations strictly dominated strategies.

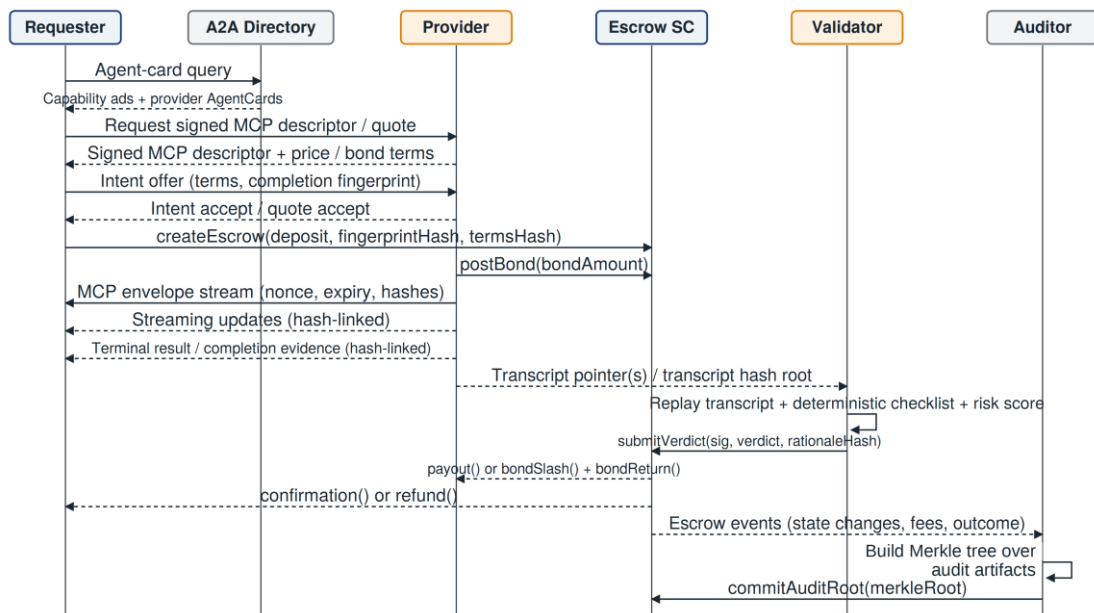


Figure 2. Canonical LLM-MCP micro-escrow sequence from discovery to settlement

2.3. Blockchain layer and smart contract

Trust bottoms out at the blockchain layer [21]. Smart contracts live here, holding funds and adjudicating outcomes based on validator input. The escrow contract itself is an automaton with five main states: created, funded, inprogress, completed, and disputed. A new escrow starts in created and waits for the requester's payment deposit. When the correct amount arrives, along with any required provider bond, the contract moves to funded and signals the provider to begin work.

Delivery triggers the next transition: as the provider starts sending context envelopes, an on-chain call moves the contract to inprogress. Everything then hinges on submitVerdict (bytes sig, bytes32 rationaleHash). A valid signature and a favorable verdict take the contract to completed, where the provider receives the deposit minus fees and the validator and auditor collect their reward splits. A failure verdict, or an expired deadline, takes it to disputed instead: the requester is refunded and the provider's bond is slashed to the treasury. Every critical event also emits an on-chain log carrying the escrow ID, the new state, and the block number, so off-chain indexing services can track escrow lifecycles with no extra machinery.

Figure 3 draws the whole machine: six states, their transitions, and how each map to the on-chain calls and settlement outcomes defined by the escrow contract. Two features stand out in the diagram. The branch point sits at inprogress, where the validator's submitVerdict sends the escrow to either completed or disputed; and both terminal branches converge into terminated once funds have been disbursed. Making the structure this explicit is what rules out ambiguous escrow outcomes under timeouts or missing verdicts; the contract stays a minimal deterministic core, and semantic interpretation stays off-chain.

View functions are kept pure. They read only contract storage or input parameters and nothing else; no external context can leak into a read. Evidence never appears raw either. The contract sees only hashes, which makes it transport-agnostic and privacy-preserving [22] and lets the audit layer verify escrow outcomes uniformly across modalities (studio logs and HTTP exchanges) as long as they produce a consistent hash trail. The hash-only model pays a second dividend: compatibility with zero-knowledge rollups [23] follows directly, and the architecture accommodates future extensions including zk-SNARK proofs of checklist satisfaction and cross-chain light escrow adapters [24].

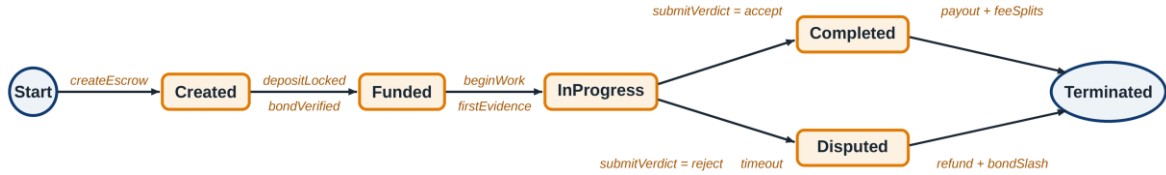


Figure 3. Micro-escrow smart contract state machine

2.4. Large language models validation and auditing

How does an LLM verdict become reproducible enough to settle money? Through a deterministic checklist. The validator evaluates a predefined set of predicates derived directly from the escrow agreement, so completion conditions exist in a structured, machine-evaluable form. Each predicate declares three things — whether it is mandatory, its relative weight, and the evidence required for evaluation — which lets hard constraints and softer quality requirements live inside a single validation framework.

Take a concrete agreement: “the temperature data must be provided at 1-minute intervals for 60 minutes, with no more than 5% missing readings”. The checklist for it might ask: “Were at least 57 temperature readings received (out of 60 expected)?”, “Is the interval between consecutive readings approximately 1 minute throughout?”, and “Do all readings fall within plausible temperature ranges?”. Every item is written as a question or an assertive statement that the evidence can answer. The validator agent then encodes the items into a prompt that constrains the LLM’s output to an easily interpreted form, typically a series of yes/no answers or one final binary decision. Under a fixed model version and fixed decoding parameters (temperature = 0), the whole evaluation is deterministic, which is what makes it reproducible in our experimental environment.

Algorithm 1 formalizes this evaluation procedure. Given a transcript T , a checklist C composed of mandatory and weighted predicates, and a risk threshold τ , the validator first reconstructs a canonical record R . Each predicate $i \in C$ is evaluated against R , yielding a binary outcome $out_i \in \{PASS, FAIL\}$. If any mandatory predicate fails, the validator immediately emits a rejection verdict. Otherwise, a scalar risk score is computed as the weighted sum of failed non-mandatory predicates. The transaction is accepted only if this aggregate risk remains below the predefined threshold τ .

Auditability without content exposure comes from the report format. The validator emits a canonical report (predicate identifiers, flags, weights, outcomes), serializes it deterministically, and hashes it with SHA-256; that rationale hash is recorded on-chain together with the binary verdict. A confidence proxy may be emitted too. It stays strictly off-chain, feeding analysis or committee weighting and nothing else. Algorithm 1 shows the complete checklist evaluation and risk-scoring process.

Algorithm 1. Deterministic checklist evaluation and risk scoring (per validator)

Input: transcript T ; checklist predicates C , where each predicate i has a mandatory flag $m_i \in \{\text{true}, \text{false}\}$ and a weight $w_i \geq 0$; risk threshold $\tau \geq 0$.
Output: $verdict \in \{\text{ACCEPT}, \text{REJECT}\}$; $rationaleHash$; optional confidence proxy (κ).
Procedure:
1. $R \leftarrow \text{ReplayDeterministically}(T)$ $\triangleright R$ is the canonical evidence record obtained by deterministic replay
2. for each predicate i in C (in a stable, predefined order): $(out_i, evidence_i) \leftarrow \text{EvaluatePredicate}(i, R)$, where $out_i \in \{\text{PASS}, \text{FAIL}\}$ $\triangleright evidence_i$ is optional supporting evidence
3. if there exists any predicate i such that $(m_i = \text{true})$ and $(out_i = \text{FAIL})$:
 $verdict \leftarrow \text{REJECT}$; else: $risk \leftarrow \sum [w_i \cdot 1(out_i = \text{FAIL})]$ over all predicates i with $m_i = \text{false}$; if $risk < \tau$: $verdict \leftarrow \text{ACCEPT}$, else $verdict \leftarrow \text{REJECT}$
4. $report \leftarrow \text{CanonicalizeReport}(\{i, m_i, w_i, out_i\} \text{ for all } i \text{ in } C)$
5. $rationaleHash \leftarrow \text{SHA-256}(report)$
6. $\kappa \leftarrow \text{ConfidenceProxy}(R, report)$ (optional)

Algorithm 1's output is the atomic decision unit of the validation layer. Over the same canonical transcript, every validator runs the checklist independently and produces a signed decision tuple. What happens next depends on deployment. A single validator submits its tuple straight to the escrow contract, and settlement follows; a committee generates multiple outputs in parallel and aggregates them into one authoritative outcome. Aggregation touches signed validator outputs only, never reinterpreting raw evidence, which preserves determinism and confines the attack surface to validator behavior.

Figure 4 turns the deterministic checklist evaluation of Algorithm 1 into a committee-based pipeline. Everything in the pipeline starts from the canonical transcript hash chain: the complete, ordered evidence log of the escrow execution. All validator agents share this chain, and it serves as the immutable reference for semantic validation.

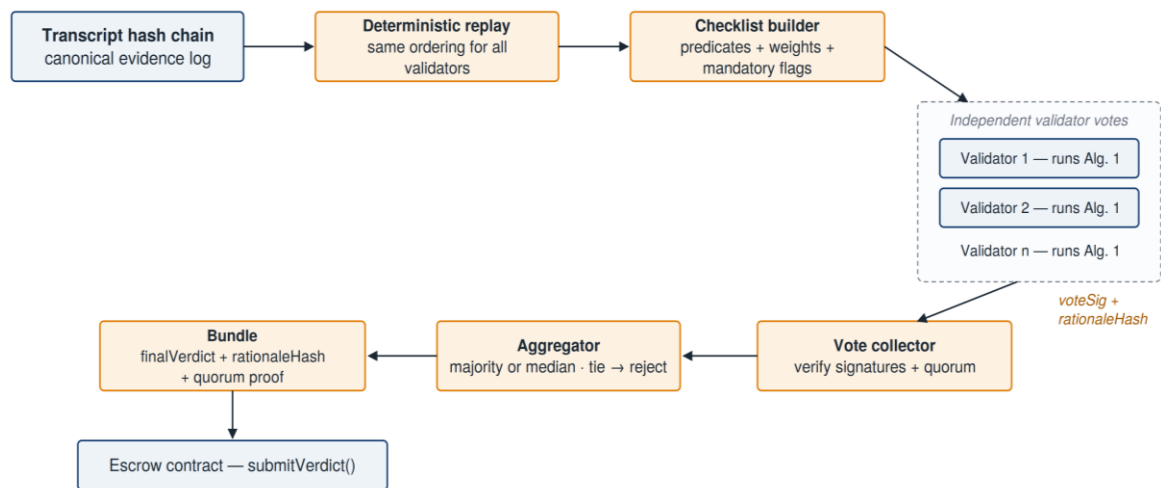


Figure 4. Committee-based verdict submission and aggregation

Replay comes first. Each validator deterministically re-executes the transcript, so all of them reconstruct the same canonical evidence record under an identical ordering of events. A checklist builder then instantiates the predicate set associated with the escrow agreement, weights and mandatory flags included. Same conditions, same configuration, for every validator. Without that, vote aggregation would mean nothing.

Execution stays local. As shown in the figure, each committee validator applies Algorithm 1 on its own and signs a vote: a binary verdict plus a rationale hash committing to the evaluation outcome. No intermediate reasoning crosses between validators, and no raw evidence does either. It is constrained this way deliberately, because cross-validator influence is precisely what would destroy statistical independence.

A vote collector gathers the signed outputs, verifies every signature, and enforces the quorum requirement. Once quorum is reached, aggregation is deterministic: a predefined rule such as majority or median selection combines the votes, and ties resolve conservatively, in favor of rejection. Out comes a compact bundle. It holds the final verdict, the associated rationale hash, and a proof that the quorum conditions were satisfied.

The bundle reaches the chain through the submitVerdict function. As illustrated in Figure 4, the contract re-evaluates no semantic conditions and inspects no execution evidence; cryptographic validity and quorum compliance are all it verifies before triggering settlement. The division holds to the end: semantic interpretation off-chain, on-chain logic minimal, deterministic, and auditable.

What Figure 4 shows, in the end, is scaling without exposure: deterministic local validation (Algorithm 1) grows into a fault-tolerant committee setting while the on-chain attack surface stays fixed. Accountability survives through signed votes and rationale hashes; individual validator failure is absorbed by aggregation over the shared canonical transcript. The cost of that resilience is bounded and predictable: every additional committee member adds one independent evaluation of the same canonical transcript, so validation effort scales with committee size rather than with the volume of evidence.

2.5. Auditing, fee adjustment, and parameter calibration

The auditor watches everything and touches nothing. It continuously samples verdict hashes, cross-links them with escrow event logs, and reconstructs a statistical view of system performance across all completed escrows: systemic governance, not a per-escrow control. Recency comes from a sliding window W of recent escrows (e.g., the last 1,024 escrows or the last 24 h), which keeps anomaly scores tied to recent behavioral trends while remaining computationally lightweight. Within the window, the auditor computes per-provider anomaly scores via a Kolmogorov–Smirnov goodness-of-fit test [25] on the distribution of latency, success rate, and other metrics compared to historical baselines. The raw material is rich: each settlement carries the provider’s response latency, the contract outcome (pass or fail), the validator’s confidence score if available, gas used, and the provider’s public key. Aggregating these attributes over the sliding window yields four key provider metrics: latency percentiles (e.g., 50th, 90th, and 99th percentile response time), success ratio (fraction of escrows passed), bond forfeiture ratio (fraction of escrows where the provider’s bond was slashed), and a distribution-drift measure from the Kolmogorov–Smirnov test. Together, the four define the provider’s operational profile. Detection then reduces to a distance question. The auditor applies a model such as a Gaussian mixture model (GMM) to compare the current profile against historical norms, and if the Mahalanobis distance of the profile from the historical cluster exceeds a threshold (e.g., $\lambda = 4.5$), the provider is flagged as anomalous.

When anomalous behavior is detected, the auditor triggers a deterministic fee adjustment. The applied fee is defined for an escrow executed by provider p as:

$$\text{feeApplied}_p = \text{feeDeclared}_p \cdot (1 + \Delta\phi_p)$$

where $\Delta\phi_p \geq 0$ is a surcharge derived from normalized risk indicators. Specifically:

$$\Delta\phi_p = \text{clip}(0, \phi_{\max}, \kappa_\ell \cdot z_\ell(p) + \kappa_d \cdot z_d(p) + \kappa_a \cdot z_a(p))$$

Here $z_\ell(p)$ is a latency deviation score (e.g., standardized drift of p_{90} or p_{99} latency relative to the provider’s historical baseline), $z_d(p)$ is a distribution-drift score (e.g., Kolmogorov–Smirnov distance on latency or outcome distributions), and $z_a(p)$ is an availability/behavior score (e.g., deviation in success ratio or unusual slash frequency). The κ coefficients control responsiveness to each signal, while $\phi_{\max}$ bounds the maximum surcharge to avoid destabilizing the marketplace. This surcharge applies only to the provider’s payout; the requester’s cost remains unaffected. Any fees collected from surcharges are redistributed to validators and auditors or burned, creating a self-correcting economic feedback loop where anomalous providers incur higher costs until their performance returns to normal.

In the simulator configuration, the auditor uses a sliding window of $W = 1,024$ recent escrows (or the last 24 h, whichever is smaller) and bounds the maximum surcharge at $\phi_{\max} = 0.30$ so that the applied fee remains dominated by the provider’s declared price. Coefficients κ_ℓ , κ_d , κ_a are selected using a benign baseline followed by a small sweep over drift and failure regimes to satisfy: i) median $\Delta\phi_p \approx 0$ and $p_{95}(\Delta\phi_p) \leq 0.05$ in benign workloads, and ii) monotonic increase under injected latency drift or abnormal failure/slash rates. As an illustrative example, for $\text{feeDeclared} = 2.5$ and normalized scores $z_\ell = 0.8$, $z_d = 0.3$, $z_a = 0.2$ with $\kappa_\ell = 0.08$, $\kappa_d = 0.05$, $\kappa_a = 0.06$, the surcharge is $\Delta\phi_p = 0.08 \cdot 0.8 + 0.05 \cdot 0.3 + 0.06 \cdot 0.2 = 0.091$, yielding $\text{feeApplied} = 2.5 \cdot (1 + 0.091) = 2.73$. The adjustment rule is therefore fully reproducible: given W , κ , and the coefficient set, feeApplied is a deterministic function of recent settlement metadata.

Slashing severity is configured separately from fee responsiveness because it represents a one-time penalty tied to confirmed failure. For a provider bond B and a slashing fraction $s \in [0, 1]$, the contract transfers sB to the treasury upon a reject outcome attributable to provider non-compliance. In practice, s is chosen so that the expected loss from cheating dominates the expected gain from a successful malicious completion (i.e., sB exceeds the provider’s expected profit on that escrow after accounting for detection probability), while remaining low enough to avoid excessive downside for benign providers under noisy evidence. The design intent is explicit: fees price monitoring and performance drift continuously, while slashing penalizes confirmed breach as a discrete one-time event.

Figure 5 assembles the complete simulation and experimental setup used for the evaluation in section 3. The pipeline runs from scenario configuration through a deterministic discrete-event simulator to the emitted artifacts and the analysis stage, with the auditor feedback loop applying the fee-adjustment and slashing parameters defined above. Every stage is seeded. Every stage replays. Feed in same configuration and identical outputs come back out, which guarantees the reproducibility of all reported results.

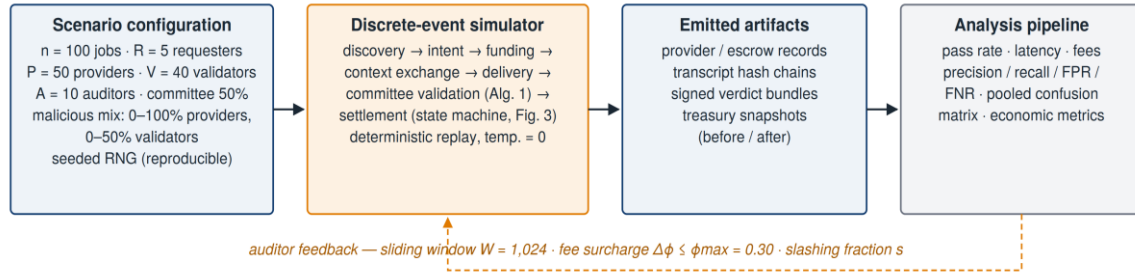


Figure 5. Simulation and experimental setup

3. RESULTS AND DISCUSSION

Four questions drive the evaluation: i) correctness and throughput under nominal conditions; ii) sensitivity to malicious provider prevalence; iii) security/latency trade-offs as committee size varies; and iv) the economic impact of slashing severity and malicious validators. Correctness, performance, robustness, and economic alignment are measured in a simulator that generates reproducible datasets together with before/after treasury snapshots. The LLM validator's decision quality gets explicit treatment too, covering precision, recall, and failure modes, and the main plots and tables are interpreted with practical implications for system tuning.

3.1. Core components

Each simulated participant is an agent with three things: operational state, performance statistics, and a signed discovery card used during capability lookup and selection. Auditability follows directly. Because the task intent, the evidence transcript hash chain, and the committee verdict bundle are all preserved, any escrow can be replayed deterministically off-chain against the published rules. The simulator seeds every random process, so outputs stay stable across runs and remain suitable for benchmark comparison. Figure 6 shows a representative provider record: emitted by the simulator, then consumed by selection, auditing, and post-hoc analysis.

```
{
  "id": 101,
  "fee": 2.5,
  "reputation": 5.0,
  "bond": 3.0,
  "latency ms": 180,
  "tasks completed": 12,
  "avg latency ms": 205,
  "registered at": "2025-09-28T12:03:40.000Z",
  "performance": {
    "total passes": 11,
    "total fails": 1,
    "avg confidence": 0.86,
    "last updated": "2025-09-28T12:10:01.000Z",
    "total pass rate": 0.9167
  },
  "a2a card": {
    "name": "Image Processing Provider",
    "role": "provider",
    "endpoint": "/agents/101",
    "capabilities": [
      {
        "type": "image processing",
        "description": "Process and analyze images"
      }
    ],
    "skills": ["image processing"],
    "authentication": {
      "type": "none", "description": null
    }
  }
}
```

Figure 6. Example provider record

3.2. Experimental scenarios

The baseline is clean by construction: $n = 100$ jobs, $P = 50$ providers, $V = 40$ validators, committee fraction = 50% (avg committee size 20), $R = 5$ requesters, $A = 10$ auditors, and not a single malicious actor. Figure 7 reports the resulting baseline outcome object, which isolates nominal bottlenecks and gives every later stress test its reference point. Correctness is perfect: $\text{pass_rate} = 1.0$. The dominant cost is validator committee inference time, not provider execution time.

Two regimes separate cleanly in the baseline. Provider execution stays sub-second at the median; committee validation stretches over several seconds on average. The gap points at where throughput lives: validator capacity (parallel committees, batching, or lighter checklists) is the lever, not provider performance under benign behavior. Economics are quiet here. Fee collection holds stable and slashing stays at zero, exactly what the absence of disputes and detected cheating should produce.

```

{
  "jobs": 100,
  "stats": {
    "passed": 100,
    "failed": 0,
    "pass_rate": 1.0,
    "avg_confidence": 0.8835,
    "median_latency_ms": 879,
    "avg_provider_latency_ms": 979.96,
    "avg_validation_time_ms": 5292.6,
    "fees_collected_delta": 319.88,
    "slashed_delta": 0
  },
  "counts": {
    "escrows": 100,
    "participants": 55,
    "avg_validator_committee_size": 20
  },
  "treasury_delta": {
    "balance_delta": 319.88,
    "fees_collected_delta": 319.88,
    "slashed_delta": 0
  }
}

```

Figure 7. Baseline outcome object

Applied fees track declared fees, but not rigidly. Figure 8 makes the relationship visible by extending the baseline to pricing: each point is a completed escrow, plotted by transaction amount, the provider's declared fee, and the fee actually applied after audit adjustments. The question the plot answers is whether applied fees merely mirror declared pricing or carry additional system-level corrections. They carry corrections. Dispersion grows at higher transaction amounts: the signature of contextual risk and monitoring overhead entering the fee computation as economic stakes rise, not of a rigid, value-independent pricing rule.

Does economic pressure erode validation quality? Figure 9 checks, reporting the empirical distribution of confidence scores across the 100 baseline escrows. The histogram is tightly concentrated: every escrow falls within a narrow band around a mean of 0.884, so the deterministic checklist returns decisive semantic judgments rather than marginal or ambiguous outcomes. No low-confidence case appears anywhere in the baseline, and that absence suggests that neither fee adjustment nor stake size systematically induces semantic uncertainty.

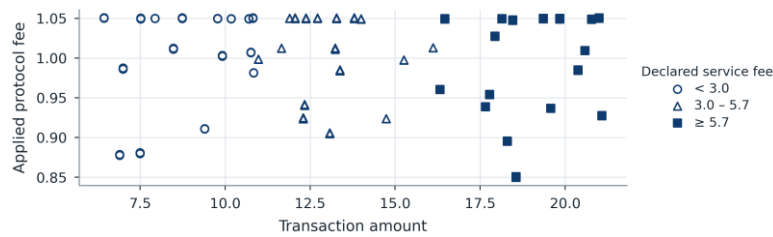


Figure 8. Amount vs declared fee vs applied fee

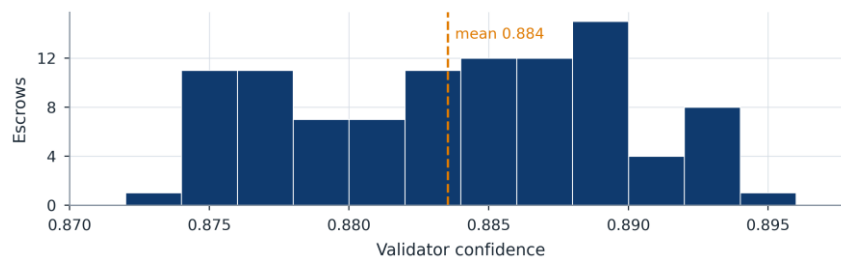


Figure 9. Distribution of validator confidence

With semantic confidence holding steady, Figure 10 connects validation outcomes back to operational performance by isolating the relationship between provider latency and the applied fee. The association is positive, exactly as the incentive design intends: providers that run slower impose greater monitoring and coordination costs, and they pay for it through higher fees. The coupling internalizes performance inefficiency economically, and it does so continuously. No coarse reputation threshold, no hard exclusion; suboptimal behavior gets discouraged through fee adjustment alone.

Taken together, Figures 8 to 10 trace a single feedback loop: semantic validation determines outcomes, outcomes shape pricing, and pricing responds to observed performance without destabilizing validation quality. Inside that loop, the semantic failures we observe split into two camps, and each camp informs checklist design. False positives (FP) appear when task specification is underspecified — tolerances that were never encoded as explicit predicates — so the remedy is tighter bounds and deterministic jitter

windows. False negatives (FN) are the mirror image: evidence that reads as semantically plausible while violating hard constraints on timestamps, ordering, or schema validity, which is why mandatory predicates short-circuit acceptance the moment one is violated. The lesson is blunt. Semantic robustness rests on predicate design, not on economic pressure or transaction scale.

Deployment guidance falls out of these observations directly. The committee fraction is the primary tuning lever, because committee inference dominates end-to-end latency: widen the committee and the honest-majority margin on which security ultimately rests widens with it, but validation time and monitoring cost grow in proportion to committee size. So, size committees against the assumed adversary share, with a safety margin, and win throughput back through parallel committee evaluation and evidence batching rather than weaker quorums. Low-value escrows get a cheaper recipe. Lighter checklists with fewer non-mandatory predicates cut validation cost, while the mandatory constraints keep the security floor intact.

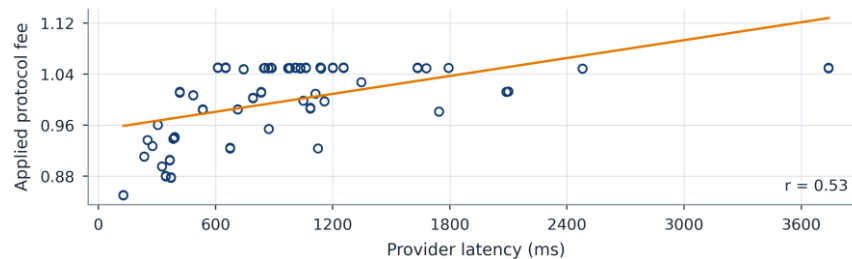


Figure 10. Provider latency vs fee applied

3.3. Validation effectiveness under adversarial conditions

Treating the validator as a binary classifier over jobs is what makes this section's assessment principled: decision accuracy, robustness boundaries, and economic side effects all become measurable under adversarial conditions. The failure-mode categories of section 3.2 carry over. Underspecified predicates and constraint violations now show up as quantified precision and recall degradation across four adversarial scenarios. One treatment covers three questions: how semantically reliable LLM validation is, how it behaves under coordinated adversaries, and how far economic incentives mitigate, or amplify, the observed failure modes.

Conventions first. A job is positive if it is malicious and negative otherwise; a validator decision of fail/reject counts as a predicted positive, and pass/accept as a predicted negative. Everything else follows: true positives (TP) are malicious jobs correctly rejected, FN malicious jobs incorrectly accepted, FP benign jobs incorrectly rejected, and true negatives (TN) benign jobs correctly accepted. One computational choice matters here. All reported metrics were computed from pooled confusion-matrix counts across runs rather than from averages of per-run ratios, because averaging ratios distorts exactly the regimes that matter most: those with vanishing numbers of malicious jobs or rejected predictions.

Recall has a direct operational reading: it is the malicious-fail rate, the fraction of malicious jobs that are correctly rejected. Precision, false positive rate (FPR), and false negative rate (FNR) are derived from the same pooled TP, FP, FN, and TN counts, and each exposes a complementary failure mode that aggregate pass rates alone keep invisible. Four metrics, one complete characterization of the validator's decision quality under each adversarial scenario.

Aggregate task-level outcomes are first reported under the different adversarial configurations, grounding the classifier metrics in operational context. Table 1 reads on two axes: pass rate is end-to-end availability (how often the system completes jobs successfully), while malicious fail rate is security preservation, the fraction of malicious jobs correctly rejected. With no adversaries, correctness is perfect and every benign job is accepted. With a small fraction of malicious providers, the validator rejects all malicious executions while benign jobs keep a high pass rate; semantic discrimination is strong. When malicious providers dominate, all jobs are rejected: security preserved, throughput sacrificed. When providers and validators are compromised together, the behavior inverts sharply. Pass rates rise again despite widespread malicious execution. That is not improved correctness; that is validator capture.

To move beyond pass rates into semantic decision quality proper, Table 2 reads as follows: recall and FNR quantify security loss, the malicious behavior that was missed, while precision and FPR quantify the availability and economic cost imposed on benign participants through incorrect rejections. Two metric degeneracies arise naturally, and both need naming. In the no-malicious baseline there are no positive instances at all, so precision and recall are undefined and reported as N/A; FPR remains meaningful there and

confirms the absence of spurious rejections. In the collapsed class cardinality regime, where all jobs are malicious, FPR is undefined instead, and although precision and recall remain mathematically defined, they lose discriminative meaning, since no trade-off between false acceptance and false rejection can be observed.

Table 1. Completion outcomes under different global agent pool compositions

Jobs	Honest provider	Honest validators	Malicious providers	Malicious validators	Pass rate	Malicious fail rate
50	50	40	0	0	1	0
50	50	40	5	0	0.9	1
50	50	40	100	0	0	1
50	5	5	50	50	0.98	0.02

All four columns are absolute agent counts. Malicious providers and validators are instantiated in addition to the honest pool, so the total population of each role is the sum of its honest and malicious columns.

Table 2. LLM validator decision quality (pooled confusion metrics)

Scenario	Precision	Recall	FPR	FNR
Baseline (no malicious)	N/A	N/A	0	N/A
Scenario 2 (5 malicious providers)	1.00	1.00	0.00	0.00
Scenario 3 (100 malicious providers)	1.00	1.00	N/A	0.00
Scenario 4 (50/50 small network)	1.00*	0.02	N/A	0.98

* Precision dominated by class prevalence; not indicative of effective discrimination.

Validator capture hides a third, subtler degeneracy. Because captured validators overwhelmingly accept jobs, rejected executions almost vanish; as the number of predicted positives approaches zero, precision stops measuring detection capability and starts echoing the base rate of malicious jobs. The apparently perfect precision reported in scenario 4 is therefore an artifact. The real signal sits elsewhere: recall collapses to 0.02 and the false-negative rate reaches 0.98, confirming that malicious executions are almost always accepted once the validator layer itself is compromised. What makes this failure mode dangerous is its shape. It manifests as false acceptance rather than denial of service: the appearance of healthy operation while security guarantees are silently violated.

Table 3 puts numbers on all four regimes, reporting validation latency, average confidence scores, fee collection totals, and slashing activity. Benign conditions expose the first structural fact: validation time dominates end-to-end latency, so the LLM committee is the primary throughput bottleneck. Mild adversarial pressure raises validation time while confidence stays high, so the added cost reflects noisier evidence rather than degraded decision quality; when malicious providers dominate, jobs are often trivially invalid, so provider latency drops while validator resources are consumed all the same. Validator capture is the regime that should worry a deployer. There, high confidence and high pass rates coexist with near-total semantic failure: once validator trust is lost, confidence aggregation alone is insufficient.

Table 3. Adversarial performance and economic metrics

Scenario	Median latency (ms)	Avg validation time (ms)	Avg confidence	Malicious validators	Fees collected	Slashed
Baseline (no malicious)	879	5292.6	0.884	0	319.9	0
Scenario 2 (5 malicious P)	771	6513.7	0.889	0	158.7	5.8
Scenario 3 (100 malicious P)	79	4708.2	0.934	0	150.6	56.6
Scenario 4 (50/50 small network)	79	7977	0.926	50	192.6	1.2

The asymmetry in these results is fundamental, not incidental. Malicious providers — even when they dominate the workload — are tolerated by validator committees; coordinated validator compromise is not. What the asymmetry exposes is the explicit trust assumption embedded in committee-based semantic validation itself, rather than any implementation artifact. Past the point where a majority of validators collude, checklist enforcement turns adversarial. Confidence aggregation cannot restore semantic correctness from there, downstream economic penalties cannot either, and the protocol carries no internal recovery path.

No amount of checklist tuning or fee sensitivity fixes this failure mode; the validator layer itself has to be hardened, and recent surveys discuss emerging patterns for secure, scalable agent–blockchain collaboration [26]. Several apply directly here. Stake- or bond-weighted voting raises the economic cost of controlling a quorum, so capture translates into direct financial loss. Enforced committee rotation and identity churn deny an attacker sustained control. Minority-report disclosure makes systematic dissent observable even when the majority colludes, which gives governance, or automated escalation, an auditable signal to act on. The deepest option follows from the checklist being deterministic: validators could prove in zero

knowledge that a public checklist was satisfied before issuing an acceptance verdict, raising the barrier to successful capture substantially.

What can economics do here? Less than one might hope, and exactly what it was designed for. Dynamic fees and slashing price monitoring effort and execution risk; correcting semantic failure was never their job. Under benign and mildly adversarial conditions the results show them behaving as intended: fees correlate with provider latency and validation effort, and slashing penalizes detected misbehavior. Increasing scrutiny raises validation time while confidence holds steady, and correctness holds with it. Economic signals modulate performance. Semantic assurance they leave untouched.

Capture breaks the alignment. Fees continue to accrue and reported confidence remains high, yet slashing collapses: malicious jobs are being accepted rather than flagged, so there is nothing left to slash. The reading is unambiguous: economic mechanisms are second-order controls whose effectiveness depends on an honest validator layer. When validators collude, no fee schedule substitutes for semantic enforcement, which is why these mechanisms must be coupled with explicit validator-side accountability.

Sensitivity to the calibration parameters is predictable rather than chaotic. Shrink the window below $W = 1,024$ and surcharges react faster to behavioral drift but amplify outcome noise; stretch it and pricing stabilizes at the cost of slower repricing once a provider degrades. The cap $\phi_{\max} = 0.30$ bounds worst-case cost inflation for requesters, keeping audit-driven adjustments a modulation on declared prices rather than their master, consistent with the bounded fee dispersion observed in Figure 8. The slashing fraction s is the delicate one. It trades deterrence against participation: expected losses from cheating must exceed expected gains, yet penalties must remain moderate enough not to expel honest providers under noisy evidence. In practice that means calibrating s jointly with the bond size.

Capture leaves fingerprints in deployment. Two signals operationalize it: sustained disagreement between minority reports and the aggregated verdict stream, or sudden structural changes in rejection rates conditional on ground-truth audit samples. When either trigger fires, the response is automatic: the protocol increases committee size, requires a higher validator bond for participation, and routes escrows to a secondary validator pool drawn from a different identity set. A measurable, automated response loop, not a purely theoretical contingency.

Established settlement mechanisms frame where this design sits. Payment channels and channel networks [12]–[15] excel at high-frequency value transfer between pre-funded counterparties, but they verify payment authorization, not task semantics, and they immobilize per-channel capital, a poor match for sporadic, many-to-many agent interactions. Authenticated data feeds [17] import external facts on-chain without judging whether delivered output satisfies the negotiated semantics. Dual-deposit escrows [18] achieve cheat-proof delivery through symmetric deposits, yet still treat disputes as binary delivery events. The niche left open is the one this design occupies: replayable checklist validation supplies graded semantic adjudication, while the escrow contract retains the economic finality of channel-style settlement.

The overall picture has two sides. Deterministic LLM-based semantic validation withstands malicious providers and degrades predictably as adversarial pressure mounts; the validator layer, by contrast, is a hard trust boundary. Making that boundary explicit and pairing it with concrete mitigations is more useful to deployers than smoothing it over. It is also what motivates future work on validator-side cryptographic and economic hardening. The results also establish a property we regard as non-negotiable: semantic correctness and economic incentive alignment are complementary, yet each is independently necessary for a trustworthy escrow system.

4. CONCLUSION

Autonomous agents can execute verifiable micro-transactions. That is the claim this study set out to test, and coupling deterministic LLM-based semantic validation with blockchain settlement bears it out: settlement is correct under benign conditions and resistant to malicious providers, while economic incentives price operational risk without degrading validation quality. The contribution we care most about is reproducibility. Semantic validation becomes replayable once transcripts, verdicts, and incentives are structurally coupled; that is what grounds machine-to-machine commerce in evidence rather than trust. The boundary of the claim is equally clear: once validators collude, semantic guarantees collapse, so committee validation remains a trust assumption, not cryptographic certainty. LLM validators are accountable, but not inherently trustless, and the distinction carries concrete deployment consequences. Future work follows directly: zero-knowledge proofs of checklist satisfaction, stake-weighted voting, enforced rotation, and minority-report mechanisms, alongside validating the architecture's scalability across heterogeneous agent deployments.

ACKNOWLEDGMENTS

The authors thank the 2IACS and IESI laboratories at ENSET Mohammedia, Hassan II University of Casablanca, for providing the research environment that supported this work.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Ayman Nait Cherif	✓	✓	✓	✓	✓	✓		✓	✓		✓			
Mohamed Youssfi		✓	✓				✓			✓		✓		
Omar Bouattane		✓	✓							✓		✓		

C : Conceptualization

M : Methodology

So : Software

Va : Validation

Fo : Formal analysis

I : Investigation

R : Resources

D : Data Curation

O : Writing - Original Draft

E : Writing - Review & Editing

Vi : Visualization

Su : Supervision

P : Project administration

Fu : Funding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data and code that support the findings of this study are openly available at <https://github.com/aymannn/micro-escrow-agents>. All simulation outputs can be regenerated deterministically from the seeded configuration described in section 2.5.

REFERENCES

- [1] N. R. Jennings, "On agent-based software engineering," *Artificial Intelligence*, vol. 117, no. 2, pp. 277–296, Mar. 2000, doi: 10.1016/S0004-3702(99)00107-1.
- [2] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent communication language," in *Third international conference on Information and knowledge management - CIKM '94*, 1994, pp. 456–463, doi: 10.1145/191246.191322.
- [3] Foundation for Intelligent Physical Agents, "FIPA ACL message structure specification," *fipa.org*, 2002. Accessed: Jul. 12, 2026. [Online]. Available: <https://www.fipa.org/specs/fipa00061/SC00061G.html>
- [4] T. Brown *et al.*, "Language models are few-shot learners," in *34th International Conference on Neural Information Processing Systems*, 2020, pp. 1877–1901.
- [5] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: interactive simulacra of human behavior," in *36th Annual ACM Symposium on User Interface Software and Technology*, Oct. 2023, pp. 1–22, doi: 10.1145/3586183.3606763.
- [6] L. Wang *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, Dec. 2024, doi: 10.1007/s11704-024-40231-1.
- [7] D. Hardt, "The OAuth 2.0 authorization framework," *rfc-editor.org*. Oct. 2012. Accessed: Jul. 12, 2026. [Online]. Available: <https://www.rfc-editor.org/info/rfc6749/>
- [8] A. N. Cherif, Y. Achir, M. Youssfi, M. Elgarej, and O. Bouattane, "Zero-knowledge proofs and OAuth 2.0 for anonymity and security in distributed systems," *The International Conference on Energy and Green Computing (ICEGC'2023)*, vol. 469, Dec. 2023, doi: 10.1051/e3sconf/202346900085.
- [9] A. Enaya, X. Fernando, and R. Kashef, "Survey of blockchain-based applications for IoT," *Applied Sciences*, vol. 15, no. 8, Apr. 2025, doi: 10.3390/app15084562.
- [10] A. Dawod, D. Georgakopoulos, P. P. Jayaraman, and A. Nirmalathas, "A survey of techniques for discovering, using, and paying for third-party IoT sensors," *Sensors*, vol. 24, no. 8, Apr. 2024, doi: 10.3390/s24082539.
- [11] M. Conti, G. Kumar, P. Nerurkar, R. Saha, and L. Vigneri, "A survey on security challenges and solutions in the IOTA," *Journal of Network and Computer Applications*, vol. 203, Jul. 2022, doi: 10.1016/j.jnca.2022.103383.
- [12] C. Decker and R. Wattenhofer, "A fast and scalable payment network with Bitcoin duplex micropayment channels," in *Stabilization, Safety, and Security of Distributed Systems*, Cham, Switzerland: Springer, 2015, pp. 3–18, doi: 10.1007/978-3-319-21741-3_1.
- [13] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of Bitcoin micropayment channel networks," *Royal Society Open Science*, vol. 5, no. 8, Aug. 2018, doi: 10.1098/rsos.180089.
- [14] G. Malavolta, P. M. Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment channel networks," in *2017 ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2017, pp. 455–471, doi: 10.1145/3133956.3134096.

- [15] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, “Sprites and state channels: payment networks that go faster than lightning,” in *Financial Cryptography and Data Security*, Cham, Switzerland: Springer, 2019, pp. 508–526, doi: 10.1007/978-3-030-32101-7_30.
- [16] M. Bartoletti and L. Pompianu, “An empirical analysis of smart contracts: platforms, applications, and design patterns,” in *Financial Cryptography and Data Security*, Cham, Switzerland: Springer, 2017, pp. 494–509, doi: 10.1007/978-3-319-70278-0_31.
- [17] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, “Town crier: an authenticated data feed for smart contracts,” in *2016 ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2016, pp. 270–282, doi: 10.1145/2976749.2978326.
- [18] A. Asgaonkar and B. Krishnamachari, “Solving the buyer and seller’s dilemma: a dual-deposit escrow smart contract for provably cheat-proof delivery and payment,” in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2019, pp. 262–267, doi: 10.1109/BLOC.2019.8751482.
- [19] M. Castro and B. Liskov, “Practical byzantine fault tolerance and proactive recovery,” *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, Nov. 2002, doi: 10.1145/571637.571640.
- [20] R. Kumaresan and I. Bentov, “How to use bitcoin to incentivize correct computations,” in *2014 ACM SIGSAC Conference on Computer and Communications Security*, Nov. 2014, pp. 30–41, doi: 10.1145/2660267.2660380.
- [21] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: a provably secure proof-of-stake blockchain protocol,” in *Advances in Cryptology – CRYPTO 2017*, Cham, Switzerland: Springer, 2017, pp. 357–388, doi: 10.1007/978-3-319-63688-7_12.
- [22] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, “DECO: liberating web data using decentralized oracles for TLS,” in *2020 ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2020, pp. 1919–1938, doi: 10.1145/3372297.3417239.
- [23] B. Parno, J. Howell, C. Gentry, and M. Raykova, “Pinocchio: nearly practical verifiable computation,” in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 238–252, doi: 10.1109/SP.2013.47.
- [24] M. Herlihy, “Atomic cross-chain swaps,” in *2018 ACM Symposium on Principles of Distributed Computing*, Jul. 2018, pp. 245–254, doi: 10.1145/3212734.3212736.
- [25] M. A. Stephens, “EDF statistics for goodness of fit and some comparisons,” *Journal of the American Statistical Association*, vol. 69, no. 347, pp. 730–737, Sep. 1974, doi: 10.1080/01621459.1974.10480196.
- [26] M. M. Karim, D. H. Van, S. Khan, Q. Qu, and Y. Kholodov, “AI agents meet blockchain: a survey on secure and scalable collaboration for multi-agents,” *Future Internet*, vol. 17, no. 2, Feb. 2025, doi: 10.3390/fi17020057.

BIOGRAPHIES OF AUTHORS



Ayman Nait Cherif    is a software engineer and Ph.D. student in Engineering Sciences at ENSET, Hassan II University of Casablanca, Morocco. His research spans blockchain systems, distributed multi-agent architectures, and privacy-preserving authentication, with recent work on CQRS architectures combined with zero-knowledge proofs for secure decision-making in multi-agent systems. In industry, he builds digital-trust and electronic-signature products, bridging applied cryptography with real-world systems. His current interests include agentic AI coordination, settlement for micro-escrow, and ZK proof systems for semantic verification. He is affiliated with the 2IACS and IESI laboratories at ENSET and actively shares research outputs and software on scholarly and developer platforms. He can be contacted at email: ayman.nait.cherif@gmail.com.



Mohamed Youssfi    is a teacher researcher in Computer Science, Parallel and Distributed Systems at the Hassan II University, ENSET Institute as well as software consultant. His researches are focused on parallel and distributed computing technologies, grid computing and middleware’s, received the B.S. degree in Mechanics in 1989 and the M.S. degree in applied mechanics in 1993 from the ENSET Institute, Mohammedia. He owns a successful YouTube channel specialized in computer engineering and software programming. He can be contacted at email: med@youssfi.net.



Omar Bouattane    has received his Ph.D. from the Hassan II University of Casablanca, in parallel computing and image processing (2001). He currently serves as a full professor in the Department of Electrical Engineering at ENSET of Mohammedia. His research topics are various and large in multiple domains of high-performance computing, parallel image processing and electrical engineering, particularly in renewable energy and smart grids. Since 2012, he was the head of the laboratory of signals, distributed systems, and artificial intelligence. Nowadays, he is the director of ENSET Institute Mohammedia since 2018. He can be contacted at email: o.bouattane@gmail.com.