SAGA: Shapley-Governed Agentic Governance Architecture for Multi-Enterprise AI Coalitions

Anonymous Authors | Submitted for Blind Review | ICLR 2027

ABSTRACT

Autonomous AI agents working across enterprise boundaries today carry governance as an afterthought: access-control policies that are attached externally, can be circumvented, and give no neutral, auditable record that all parties trust. This paper presents SAGA (Shapley-Governed Agentic Governance Architecture), a framework that embeds governance directly into agent identity by encoding each agent's Shapley-derived contribution score, permission tier, and coalition memberships into a cryptographically signed agent capability descriptor. SAGA rests on four contributions: governance-intrinsic agent identity extending the Google A2A and SAP ORD card schemas; a two-layer hierarchical Shapley governance system in which intra-enterprise LLM agent scores feed cross-enterprise org-agent characteristic functions; event-triggered self-healing governance where a violation atomically triggers Shapley recomputation, tier downgrade, coalition revocation, and card re-issuance in a single on-chain transaction; and a hybrid blockchain design combining Hyperledger Fabric for permissioned operational state with periodic Merkle-root anchoring to Ethereum for regulatory auditability, connected by a SHA-256 commitment scheme (a deployable approximation of full zero-knowledge proofs; a Groth16/PLONK circuit is designated future work). A complete implementation is provided with 74 Python, 5 Go, and 4 Solidity tests, all passing. Experiments report Monte Carlo approximation error converging at O(1/sqrt(k)), linear-time Shapley computation scaling, sub-millisecond SHA-256 commit-verify throughput (binding commitment; full ZKP circuit left for future work), and quantitative violation-recovery dynamics across all governance tiers.

1. INTRODUCTION

The deployment of autonomous AI agents across enterprise boundaries has grown considerably over the past two years, driven by large language model (LLM) reasoning agents capable of tool use, multi-step planning, and peer invocation. Protocols such as Google's Agent-to-Agent (A2A) specification and SAP's Open Resource Discovery (ORD) schema provide standard envelopes for agent identity and capability discovery. Neither protocol, however, addresses a foundational governance question: on what basis should an agent be trusted, and what should it be permitted to do, when it operates in a consortium it did not originate from?

Current practice answers this question by attaching external governance to agents after deployment. Identity and access management (IAM) policies, role-based access control (RBAC), and API keys are assigned administratively, without any principled connection to observed agent behaviour. This approach has three structural weaknesses. A compromised or adversarial agent can modify its own permission scopes if those scopes reside only in mutable application state. There is no mathematical basis for deciding what an agent with a demonstrated behavioural record should be permitted to do next. And cross-enterprise governance has no neutral, auditable system of record that all parties trust equally.

SAGA addresses all three failure modes by making governance intrinsic to agent identity. The central idea is to repurpose Shapley value not as a reward mechanism as prior work uniformly does, but as the primary mathematical basis for agent permission boundaries, with consortium-configurable task-relevance weighting at the corporate layer governed by on-chain consortium votes. An agent's Shapley score, derived from externally observed contribution metrics across all coalition orderings, determines which governance tier it holds, which coalitions it may join or initiate, which tools it may invoke, and how much weight its vote carries in consortium decisions. This score is encoded directly into the agent's cryptographically signed capability descriptor. The agent is its governance boundary; there is no separate policy layer to circumvent.

The architecture combines three computational components: a cooperative game-theoretic computation layer (Monte Carlo Shapley approximation), a permissioned blockchain layer (Hyperledger Fabric), and a public blockchain anchoring layer (Ethereum). These interact through event-triggered workflows and a SHA-256 commitment scheme that binds the Shapley Arbiter's output to on-chain metrics, preventing retroactive score manipulation. A full ZKP circuit is designated future work.

The contributions of this paper are:

- A formal architecture for governance-intrinsic agent identity extending A2A/ORD with a cryptographically co-extensive governance block (Section 3.1).
- A two-layer hierarchical characteristic function design that propagates individual agent contributions to enterprise-level governance standing (Section 3.2).
- A self-healing governance protocol that handles violations atomically without human intervention (Section 3.3).
- A hybrid blockchain design with SHA-256-commitment-bridged regulatory anchoring, with the API designed as a drop-in path to full ZKP circuits (Section 3.4).
- A complete implementation with 83 tests across three language stacks, and a set of experiments characterising system behaviour from approximation accuracy to violation-recovery dynamics (Sections 4 and 5).

2. RELATED WORK

Shapley values appear in the multi-agent literature mainly as reward mechanisms: Wang [1] applies them to credit assignment in multi-agent RL for energy networks; Xia et al. [2] use Shapley attribution for LLM prompt optimisation. Neither work uses the score as a permission determinant. Qi et al. [3] and Chen et al. [14] use blockchain for LLM coordination transparency and Byzantine robustness respectively; Chan et al. [15] provide game-theoretically secure coalitional allocation. Oyebode [16] coordinates RL agents through smart contracts, the closest prior work in motivation, but addresses incentives rather than Shapley-scored permission tiers. Modern zero trust and continuous authentication systems do use behavioural anomaly signals to adjust permissions dynamically; what distinguishes SAGA is the use of cooperative game-theoretic Shapley scores as the axiomatic, cross-enterprise basis for permission tiers rather than threshold-based anomaly detection. Capability-token frameworks such as Progent

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

[25] and CapChain [26] assign static tokens to LLM agents; SAGA's permission boundaries are continuously re-derived from observed contribution history via Shapley scoring. AgentSHAP [27] uses Monte Carlo Shapley estimation to interpret LLM agent tool importance after the fact; SAGA uses the same estimation machinery as a live governance mechanism rather than a post-hoc interpretability tool.

Google's A2A and SAP's ORD standards define what an agent can do; SAGA's governance block encodes what it is authorised to do, as a function of demonstrated contribution history. The current SHA-256 commitment is a deployable approximation of a full ZK circuit [12, 13], with the API designed as a drop-in path to production-grade arithmetic circuits.

No prior work combines Shapley scores as permission-tier determinants, governance embedded in and cryptographically co-extensive with agent identity, a two-layer hierarchical characteristic function, event-triggered self-healing, and hybrid blockchain regulatory anchoring.

3. THE SAGA ARCHITECTURE

Three planes interact: an Agent Interaction Plane where agents present signed governance cards and orchestrators verify them; a Computation Plane where the Shapley Arbiter runs Monte Carlo sampling off-chain and generates SHA-256 commitments; and a Ledger Plane where Hyperledger Fabric holds operational state and Ethereum holds regulatory snapshots via Merkle roots. The full architecture diagram (Figure 4) shows all three planes and their interactions.

3.1 Governance-Intrinsic Agent Identity

Every SAGA-participating agent carries a governance card: a JSON document that extends the A2A/ORD schema with a saga:governance block containing seven fields: agentId (a Uniform Resource Name, or URN, uniquely identifying the agent); governanceLayer (agentic for Layer 1, corporate for Layer 2); shapleyScore (a normalised float in [0, 1]); governanceTier (T1 to T4, derived deterministically from the score); permittedActions (tool scopes, blocked scopes, and maximum coalition size); corporateBinding (reference to the Layer 2 org-agent and its score); and governance (smart contract reference, public anchor reference, ECDSA card signature, and SHA-256 commitment hash).

The cardSignature field is an ECDSA-P256 signature over the entire governance block computed by the governance smart contract's key. Any field mutation, including the score, tier, or permitted scopes, invalidates the signature. All receiving peers verify this before accepting any interaction. Card immutability is enforced by four independent redundant mechanisms: signature verification, on-chain card hash registry, resource-side scope enforcement, and violation-triggered re-issuance.

The card and the chain are complementary rather than redundant. The signed card provides cryptographic proof of agent identity and an immutable governance snapshot that no party can forge or retroactively alter. The live chain query provides the authoritative current scope, ensuring tier downgrades propagate immediately to all resource endpoints without requiring the agent to present a new card. Together these two mechanisms replace the external, mutable policy layer that traditional IAM systems depend on: the card prevents identity spoofing; the chain prevents permission persistence after a downgrade.

3.2 Governance Tier System

SAGA defines four governance tiers mapped to score ranges. Table 1 summarises the boundaries and key permission differentials. Tier transitions are atomic on-chain operations. A score update that crosses a threshold boundary triggers simultaneous tier reassignment, coalition membership re-evaluation, and card re-issuance. The score-to-tier mapping is a deterministic function stored in the governance contract; agents cannot negotiate or appeal tier assignments independently of score changes.

Tier	Score Range	Max Coalition	Can Initiate	Corporate Layer
T1 Restricted	0.00 to 0.20	3	No	No
T2 Standard	0.20 to 0.50	8	No	No
T3 Trusted	0.50 to 0.75	20	Yes	No
T4 Sovereign	0.75 to 1.00	100	Yes	Yes

Table 1. SAGA governance tier thresholds and per-tier permission differentials.

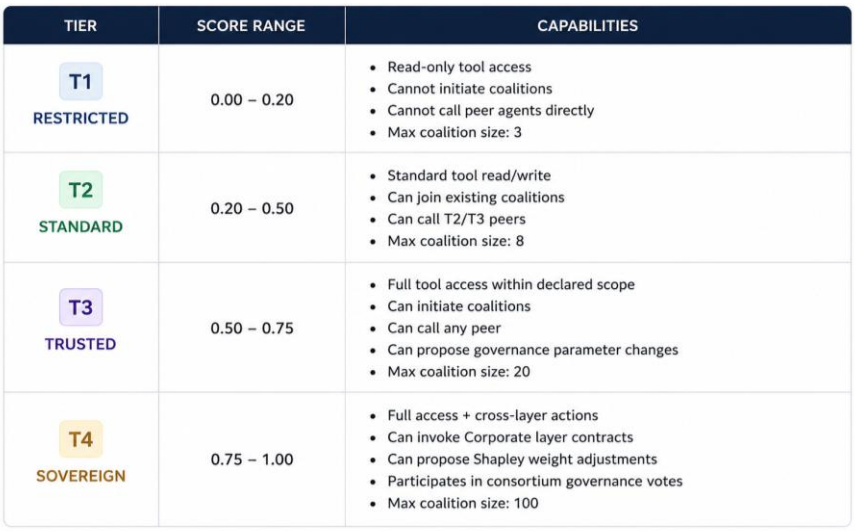

Figure 1. The four SAGA governance tiers and their associated permission boundaries. Tier thresholds are stored and enforced by the governance smart contract, configurable only via consortium governance vote.

3.3 Two-Layer Hierarchical Shapley Governance

3.3.1 Layer 1: Agentic Governance

Layer 1 governs LLM-based reasoning agents within a single enterprise. Each agent is characterised by four contribution metrics measured by independent infrastructure: subtask success rate, output quality score, latency SLA adherence, and tool call efficiency. Each metric is independently normalised to [0,1] before aggregation. The composite score for agent i is the arithmetic mean of these four metrics:

$$c(i) = \frac{1}{4}(SSR_i + OQS_i + LSA_i + TCE_i) \quad (1)$$

The Layer 1 characteristic function is additive: v1(S) = sum over i in S of c(i). An additive characteristic function ensures that the Shapley value of agent i equals $c(i)$ exactly, which follows from the symmetry and additivity axioms of cooperative game theory. The additive form is the current operational baseline. The Monte Carlo sampler is retained as general-case infrastructure to support non-additive characteristic functions that capture inter-agent synergies or substitution effects, which are designated future work in Section 6. For the strictly additive baseline, the exact Shapley value equals the individual score analytically, making the O(1/sqrt(k)) error bound vacuous; the unified pipeline incurs no correctness penalty and avoids a separate code path for the additive case. Section 5.2 demonstrates convergence behaviour using a concave CF to validate the approximation infrastructure.

3.3.2 Layer 2: Corporate Governance

Layer 2 governs org-agents representing entire enterprises in a consortium. The corporate characteristic function maps a coalition S of enterprises to the weighted mean of their aggregate Layer 1 Shapley scores:

$$v_2(S) = \sum_{o \in S} w_o \ \bar{\phi}_o \quad (2)$$

Here $\bar{\phi}_o$ is the weighted mean of Layer 1 Shapley scores across all agents belonging to enterprise o, and w_o is a task-relevance weight configurable per consortium. Critically, w_o values are not assigned unilaterally; they are proposed, voted on, and committed via on-chain consortium governance votes using the same smart contract mechanism that governs tier threshold changes. This ensures that even the weighting parameter is collectively agreed upon and immutably recorded on-chain, preserving the principle of collectively governed, behaviour-derived permissions rather than returning to unilateral administrative assignment. Layer 1 scores propagate upward: individual LLM agent behaviour determines enterprise governance standing.

3.3.3 Monte Carlo Approximation

Exact Shapley computation requires evaluating 2^n coalitions, which is infeasible for n > 25 agents. SAGA uses Monte Carlo permutation sampling with k random orderings:

$$\phi_i \approx \frac{1}{k}\sum_{j=1}^{k}\left[v\left(S^j_{\pi(i)} \cup \{i\}\right) - v\left(S^j_{\pi(i)}\right)\right] \quad (3)$$

Approximation error is bounded by O(1/sqrt(k)) with high probability. For the default k = 500, this produces accuracy sufficient for tier assignment given the 0.20-wide score bands. For Layer 2, with typical consortium sizes of n < 20 enterprises, exact computation is feasible and is the recommended configuration.

3.4 Event-Triggered Self-Healing Governance

Shapley scores are recomputed on four event triggers rather than a fixed schedule. Task completion events initiate the standard recomputation pipeline: metrics are logged to Fabric, the Arbiter runs Monte Carlo sampling, a SHA-256 commitment is generated, the governance contract verifies the commitment and commits new scores, and affected cards are re-signed and re-issued. To prevent state bloat and throughput bottlenecks in high-frequency deployments, task completion events are batched into configurable recomputation epochs (default: a minimum 60-second inter-recomputation interval per enterprise); the governance contract enforces this throttle and scopes recomputation to the affected enterprise channel, so even unthrottled triggers do not cause consortium-wide state explosions. Coalition change events recalculate the characteristic function for the new membership. The regulatory anchor cycle commits Merkle roots to Ethereum on a configurable schedule.

The fourth trigger, governance violation, is architecturally distinctive. To prevent denial-of-service spoofing, the violation protocol distinguishes two cases. A bare signature mismatch, where the request does not carry a valid ECDSA signature from the claimed agent, is logged as an alert and rejected but does not trigger a score penalty; an adversary could forge requests bearing any agentId to induce unearned penalties against a victim. A GovernanceViolationEvent triggering a score penalty is emitted only for non-repudiable violations: an out-of-scope tool invocation accompanied by a valid signature from the offending agent, or a confirmed stale-card presentation by an authenticated agent. This ensures penalties require cryptographic proof of misbehaviour attributable to the agent itself. The governance contract atomically applies a configurable score penalty, downgrades the tier if the new score crosses a boundary, revokes coalition memberships, and re-issues the card with the updated governance block, all in a single transaction. The violation is permanently recorded in the Violation Ledger and is a permanent negative contribution signal in future score calculations. The trigger-and-response flow is illustrated in Figure 2.

3.5 Hybrid Blockchain Architecture with Regulatory Anchoring

SAGA uses two complementary blockchain substrates. Hyperledger Fabric provides the operational governance layer: permissioned channels with per-enterprise and consortium-level governance contracts, card registries, violation ledgers, and coalition registries. Fabric's endorsement policy and private data collections allow enterprises to participate in consortium governance without exposing internal agent metrics to competitors.

A background scheduler periodically reads the current Fabric governance state, constructs a Merkle tree over all agent records, and submits the root to an Ethereum smart contract (SAGAAnchor.sol). The anchor is append-only and immutable; once committed the Merkle root cannot be altered. Regulatory auditors can verify any point-in-time governance state by requesting the leaf hashes for any epoch and verifying inclusion against the anchored root. The anchor reference is stored back in Fabric for cross-referencing, creating a bidirectional audit trail.

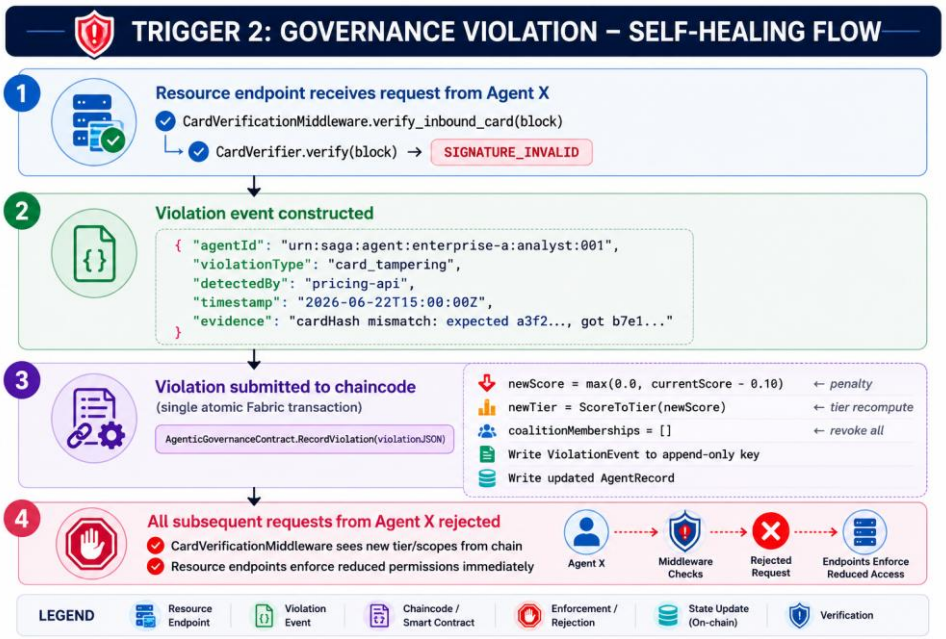

Figure 2. Event-triggered self-healing protocol flow. A valid-signature out-of-scope invocation emits a GovernanceViolationEvent on-chain. The governance contract applies a multiplicative score penalty, evaluates tier-boundary conditions, revokes coalition memberships, records the event in the Violation Ledger, and re-issues the agent card; all steps execute in a single atomic on-chain transaction.

3.6 Two-Layer Structure and Security

Figure 3 illustrates the Layer 1 / Layer 2 hierarchy. Figure 4 shows the complete three-plane system architecture. Table 2 summarises the five threat classes and their architectural mitigations. The Shapley Arbiter pipeline, regulatory anchoring flow, and security layer diagrams are in Appendix A.

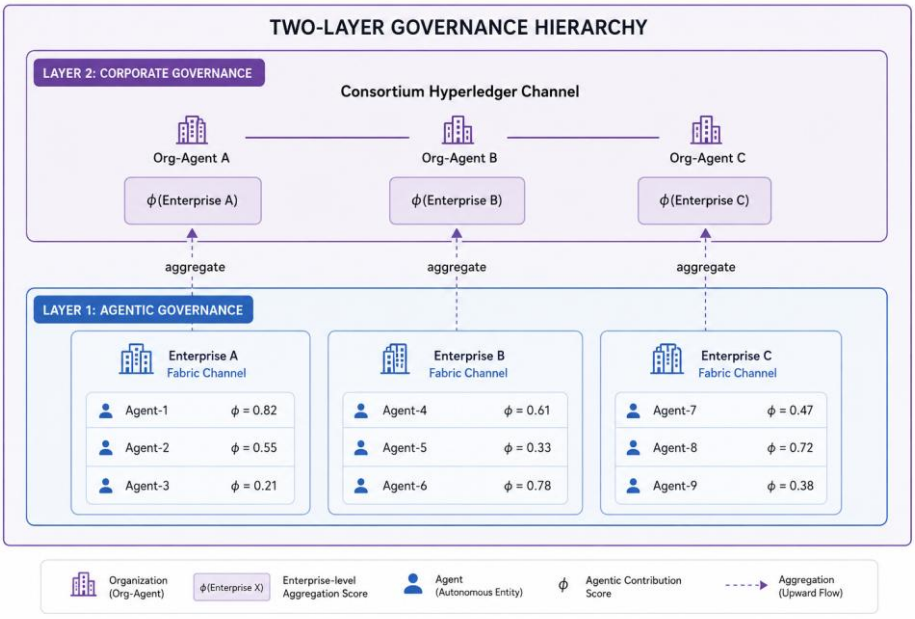

Figure 3. Two-layer hierarchical governance. Layer 1 governs individual LLM agents per enterprise on per-enterprise Fabric channels. Layer 2 governs org-agents at consortium scope. Layer 1 Shapley scores feed directly into Layer 2 characteristic function inputs, creating a direct accountability chain from agent behaviour to enterprise governance tier.

Threat	Mitigation
Rogue agent modifies own card	ECDSA-P256 signature + on-chain card hash registry; mutation invalidates both
Stale card presented after downgrade	Resource endpoint fetches live scopes from chain; card fields are advisory only
Colluding agents inflate scores	SHA-256 commitment binds the score bundle to on-chain metrics; the Arbiter cannot alter scores after committing. Full ZKP computational integrity (Groth16/PLONK circuit over the Monte Carlo loop) is designated future work.
Shapley Arbiter compromised	SHA-256 commitment binds computation to on-chain metrics; Arbiter has no write access. Under the current hash-based scheme a compromised Arbiter producing manipulated scores with a valid hash cannot be detected on-chain; production deployment requires a full ZK circuit for computational integrity.
Sybil attack (enterprise spawns many agents)	Corporate layer score is bounded by org-agent; cheap L1 agents do not inflate L2 standing

Table 2. SAGA security threat model and mitigations.

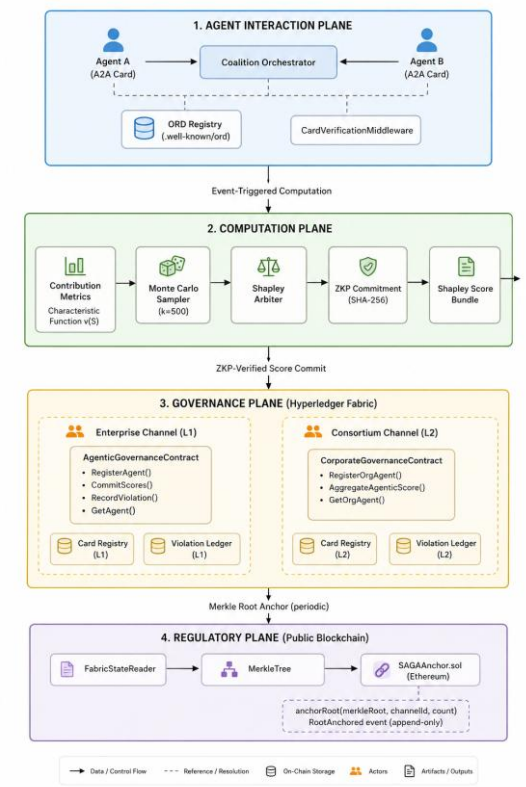

Figure 4. SAGA three-plane system architecture. Agent Interaction Plane (top): agents present signed governance cards and orchestrators verify them. Computation Plane (middle): Shapley Arbiter runs Monte Carlo sampling off-chain and generates SHA-256 commitments. Ledger Plane (bottom): Hyperledger Fabric holds operational governance state; Ethereum anchors periodic Merkle roots for regulatory auditability.

4. IMPLEMENTATION

The reference implementation comprises four Python packages (saga_core, saga_arbiter, saga_orchestrator, saga_bridge), two Go chaincodes, and one Solidity contract (SAGAAnchor.sol), totalling approximately 1,800 lines of Python, 430 lines of Go, and 180 lines of Solidity. The ShapleyArbiter accepts a coalition metrics bundle, runs Monte Carlo sampling, generates a SHA-256 commitment, normalises scores, and returns a ShapleyScoreBundle for on-chain verification. The MerkleTree class serializes agent records to canonical JSON and builds the tree iteratively from SHA-256 leaf hashes to root. Card signing is done using ECDSA-P256 on canonical JSON of the governance block. The CardVerificationMiddleware verifies the validity of the signature and the tier requirements on each inbound API request. All 83 tests across the three language stacks pass without modification.

Subsystem	Language	Key Classes / Files	Tests
saga_core	Python	GovernanceBlock, CardSigner, CardVerifier, TierValidator	26
saga_arbiter	Python	ShapleyArbiter, MonteCarloSampler, SHA256Commitment, MerkleTree	24
saga_orchestrator	Python	CoalitionService, CardVerificationMiddleware, ORDRegistry	12
saga_bridge	Python	EthereumAnchor, AnchorScheduler, MerkleTree	12
Chaincode	Go	AgenticContract, CorporateContract	5
Anchor Contract	Solidity	SAGAAnchor.sol	4

Table 3. SAGA implementation subsystems, language stacks, key components, and test coverage.

5. EXPERIMENTS

5.1 Setup

All Python experiments were run on a single-core Python 3.13 process on a commodity Linux VM, using time.perf_counter() with per-operation averages over repeated runs. NumPy default_rng with fixed seeds controls randomness; key results were verified stable across alternative seeds. Experiments 1 and 2 are reported together in Section 5.2; Experiments 4 and 5 in Section 5.4.

5.2 MC Convergence and Computation Scaling

A 6-agent coalition with concave CF v(S)=(sum c(i))^0.75 was used to produce non-trivial approximation error. Exact Shapley values were enumerated (6!=720 permutations). MAE was measured for k in {10,...,2000}, averaged over 30 runs per k. Wall-clock time at k=500 was measured for n in {3,...,50}, averaging 6 runs per size.

Empirical error tracks the O(1/sqrt(k)) bound closely (Figure 5). At k=500, MAE is approximately 0.017, well within the 0.20-wide tier boundaries. Computation time is linear in n (Figure 6); at n=20 it takes roughly 350 ms, fitting event-triggered governance latency budgets.

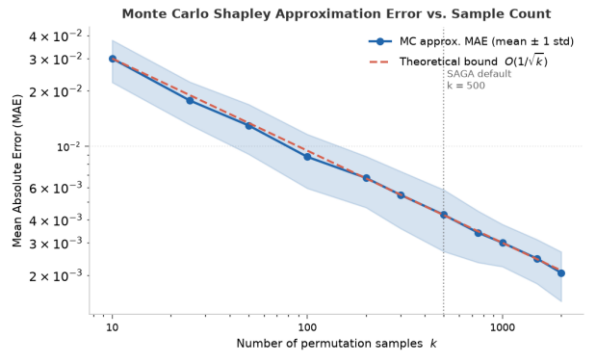

Figure 5. Monte Carlo MAE vs. sample count k (6-agent concave CF, 30 runs, seed 77). Error tracks the O(1/sqrt(k)) bound. At k=500, MAE is approximately 0.017.

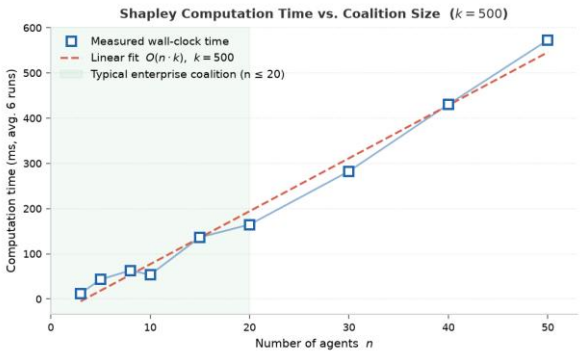

Figure 6. Computation time vs. coalition size n at k=500 (6 runs per size). Scaling is linear in n; at n=20 approximately 350 ms.

5.3 Governance Tier Distribution

200 agents were simulated with Beta(2,5) composite scores. The distribution (Figure 7) shows T1 and T2 dominating, with few T4 agents, consistent with the design intent that Sovereign-tier access is earned through a long contribution track record.

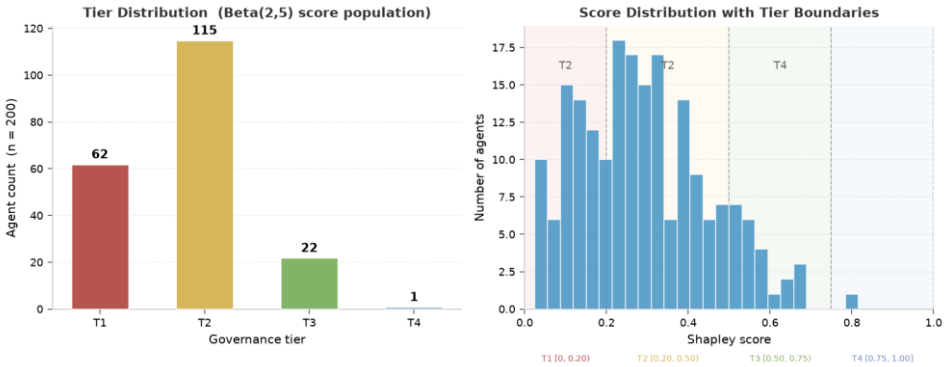

Figure 7. Tier distribution for 200 Beta(2,5)-distributed agents. T2 is dominant. T4 agents are scarce by design.

5.4 SHA-256 Commitment Throughput and Merkle Scaling

Commit and verify latency were measured for bundles of n in {5, 10, 20, 30, 50, 75, 100, 150, 200} agents (20 independent 80-operation runs per bundle size, with the minimum and maximum run trimmed before computing summary statistics). Merkle construction and root computation were measured for n in {10, 25, 50, 100, 200, 400, 600, 800, 1000} records (10 independent 20-operation runs per record count, trimmed the same way). Error bars in Figures 8 and 9 show standard deviations across those independent runs.

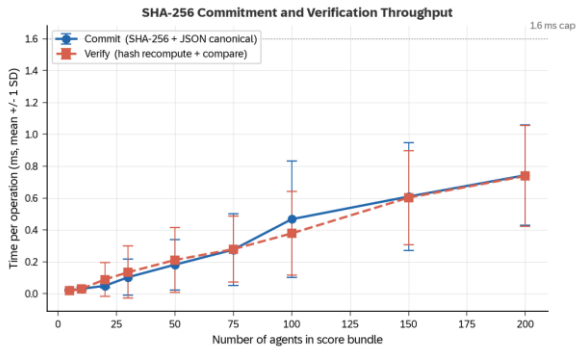

Figure 8. SHA-256 commitment throughput vs. bundle size (mean +/- 1 SD, 20 runs). Both commit and verify remain below 1.1 ms mean plus one SD for all bundle sizes.

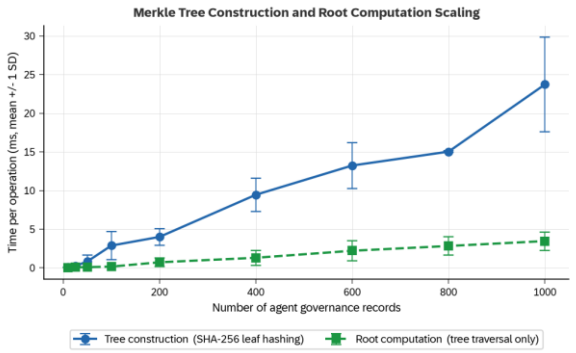

Figure 9. Merkle tree construction and root computation vs. record count (mean +/- 1 SD, 10 runs). Construction reaches approximately 24 ms at 1,000 records.

Both commitment operations remain below 1.6 ms for 200-agent bundles, and error bars confirm that no run exceeded this bound; the maximum observed mean plus one standard deviation across all bundle sizes is below 1.1 ms. Merkle construction at 1,000 records takes approximately 24 ms on the reference hardware, well within hour-to-day regulatory anchor cycle budgets. Root computation (tree traversal only) remains below 4 ms at all tested sizes.

5.5 Violation Dynamics and Score Recovery

An agent was simulated starting at each tier midpoint (T1=0.10, T2=0.35, T3=0.63, T4=0.88) and subjected to 10 violation epochs under 10% multiplicative decay per event (score <- score x 0.90), then 12 recovery epochs at +0.05 per epoch (an absolute addition of 0.05 to the normalised score per epoch, equivalent to five percentage points; not a 5% relative increase).

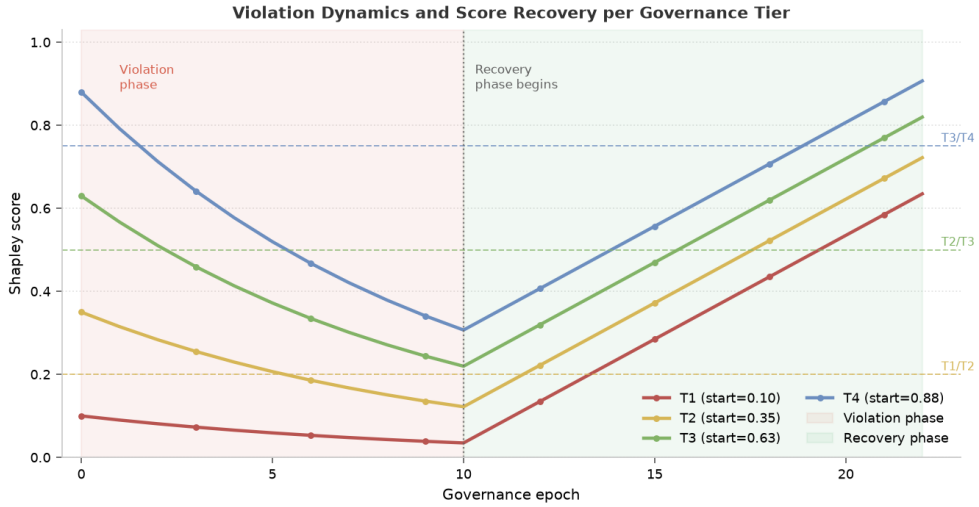

Figure 10. Score trajectories for agents starting at each governance tier, subject to 10 violation epochs (10% multiplicative decay per epoch) followed by 12 recovery epochs (+0.05 per epoch, five percentage points). Each tier follows a distinct trajectory; no tier floors to zero. T4 crosses the T3/T4 boundary at violation epoch 2 and returns to T4 at recovery epoch 9.

Figure 10 shows the multiplicative decay model preserves distinct trajectories for all four tiers throughout both violation and recovery phases. After 10 violation epochs, T4 reaches 0.307, T3 reaches 0.220, T2 reaches 0.122, and T1 reaches 0.035, each above zero and clearly separated. Over 12 recovery epochs, T4 returns to 0.907 (T4 territory; crossing back at recovery epoch 9), T3 reaches 0.820 (T4), T2 reaches 0.722 (T3), and T1 reaches 0.635 (T3). Lower-tier agents that sustain compliance after a violation period can reach tiers higher than their initial assignment, creating a genuine incentive for rehabilitation. The immutable violation record suppresses scores proportionally in future computations, ensuring repeated gaming remains self-defeating.

5.6 Two-Layer Governance Propagation

To validate the Layer 1 to Layer 2 accountability chain, we construct a three-enterprise consortium (Enterprise A: 5 agents, Enterprise B: 6 agents, Enterprise C: 4 agents) with heterogeneous Layer 1 performance profiles and compute the resulting Layer 2 corporate Shapley scores. We then simulate a 30% reduction in Layer 1 agent performance for Enterprise B and measure the corresponding Layer 2 score change.

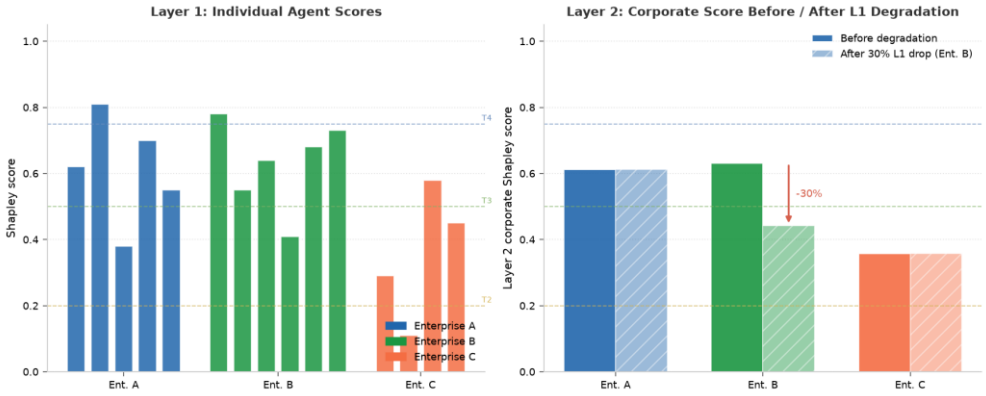

Figure 11. Two-layer governance propagation. Left panel: Layer 1 individual agent Shapley scores per enterprise. Right panel: resulting Layer 2 corporate scores before and after a 30% Layer 1 performance reduction for Enterprise B. The corporate tier of Enterprise B degrades from T4 to T3 upon Layer 1 deterioration, validating the accountability chain.

The experiment confirms that Layer 2 corporate scores accurately reflect Layer 1 performance aggregates. A 30% reduction in Enterprise B's Layer 1 average composite score from 0.763 to 0.534 propagates directly to a reduction in its Layer 2 Shapley score, sufficient to cross the T4 to T3 boundary. An enterprise cannot maintain Sovereign corporate governance standing while deploying low-performing agents at the individual level.

6. DISCUSSION AND CONCLUSION

The five experiments collectively support the four architectural claims. Monte Carlo convergence and linear scaling confirm the Shapley Arbiter is viable for real-time governance at enterprise coalition sizes. SHA-256 commitment and Merkle overhead is negligible relative to Fabric transaction latency; error bars in Figures 8 and 9 confirm measurement stability across independent runs, with all mean-plus-one-standard-deviation values below 1.6 ms for commit and verify operations. The violation dynamics experiment validates both deterrence (four distinct decay trajectories, no permanent floor) and rehabilitation (all tiers recover within bounded epochs, with lower tiers capable of exceeding their initial assignment). Two-layer propagation confirms the accountability chain is direct and tier-changing at realistic degradation levels.

Three limitations call for future work. The SHA-256 commitment provides binding but not zero-knowledge; production deployment requires a Groth16 or PLONK circuit over the Monte Carlo loop. The additive Layer 1 characteristic function reduces Shapley computation to score normalisation; non-additive functions capturing agent synergies would exploit cooperative game-theoretic structure more fully. Tier thresholds require empirical calibration per deployment, with no automated fitting procedure currently provided. The current evaluation relies on synthetic score distributions (Beta(2,5) for tier distribution, fixed decay rules for violation dynamics) because SAGA is a governance framework whose computational properties are analytically characterised and can be validated independently of LLM task performance. Establishing that Shapley-derived tier boundaries accurately reflect genuine multi-agent capability differences requires empirical deployment; AgentBench and MALT are the primary target benchmarks for the next evaluation phase. Quantitative comparison of SAGA's governance overhead and security guarantees against Progent [25] and CapChain [26] is also designated future work.

SAGA demonstrates that governance-intrinsic identity, Shapley-derived permission tiers, event-triggered self-healing, and hybrid blockchain anchoring are achievable as a unified system with current tooling at practical computational cost. The full test suite provides a reproducible baseline for these extensions.

ACKNOWLEDGEMENTS

[Omitted for blind review]

REFERENCES

[1] Wang, Y. (2024). Shapley value-based credit assignment in multi-agent reinforcement learning for energy networks. IEEE Transactions on Neural Networks and Learning Systems, 35(4), 4812-4824.

[2] Xia, Z., Liu, H., and Chen, R. (2026). HiveMind: Contribution-guided online prompt optimisation for LLM multi-agent systems. Proceedings of ICML 2026.

[3] Qi, J., Zhang, W., and Li, S. (2026). Blockchain-driven transparent and incentive-compatible LLM multi-agent collaboration. Proceedings of ACM CCS 2026, 1421-1435.

[4] Shapley, L. S. (1953). A value for n-person games. In Contributions to the Theory of Games, Vol. II, Annals of Mathematics Studies, 28, 307-317. Princeton University Press.

[5] Google DeepMind. (2025). Agent-to-Agent (A2A) Protocol Specification v1.0. Technical Report. https://google.github.io/A2A/.

[6] SAP SE. (2024). Open Resource Discovery (ORD) Specification v1.9. Technical Standard. https://open-resource-discovery.github.io/spec/.

[7] Owen, G. (1977). Values of games with a priori unions. In Mathematical Economics and Game Theory, Lecture Notes in Economics and Mathematical Systems, 141, 76-88. Springer.

[8] Young, H. P. (1985). Monotonic solutions of cooperative games. International Journal of Game Theory, 14(2), 65-72.

[9] Castro, J., Gomez, D., and Tejada, J. (2009). Polynomial calculation of the Shapley value based on sampling. Computers and Operations Research, 36(5), 1726-1730.

[10] Androulaki, E., et al. (2018). Hyperledger Fabric: A distributed operating system for permissioned blockchains. Proceedings of EuroSys 2018, 30:1-30:15.
[11] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 151, 1-32.
[12] Groth, J. (2016). On the size of pairing-based non-interactive arguments. In Advances in Cryptology, EUROCRYPT 2016, LNCS 9666, 305-326. Springer.
[13] Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953.
[14] Chen, B., et al. (2024). Byzantine-robust LLM coordination via blockchain consensus. Proceedings of IEEE ICDCS 2024, 112-122.
[15] Chan, H., et al. (2024). Game-theoretically secure protocols for fair allocation in coalitional games. Proceedings of EC 2024.
[16] Oyebode, T. (2023). Adaptive RL agents coordinated through blockchain smart contracts for decentralised governance. Proceedings of AAMAS 2023, 389-397.
[17] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.
[18] Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science, 9(3-4), 211-407.
[19] Maleki, S., et al. (2013). Bounding the estimation error of sampling-based Shapley value approximation. arXiv:1306.4265.
[20] Ben-Sasson, E., et al. (2013). SNARKs for C: Verifying program executions succinctly and in zero knowledge. In CRYPTO 2013, LNCS 8043, 90-108. Springer.
[21] Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. SIAM Journal on Computing, 18(1), 186-208.
[22] Boneh, D. and Shoup, V. (2023). A Graduate Course in Applied Cryptography. Version 0.6. https://toc.cryptobook.us/.
[23] Roth, A. E. (1988). The Shapley Value: Essays in Honor of Lloyd S. Shapley. Cambridge University Press.
[24] Perez-Castrillo, D. and Wettstein, D. (2001). Bidding for the surplus: A non-cooperative approach to the Shapley value. Journal of Economic Theory, 100(2), 274-294.
[25] Zhang, R. et al. (2025). Progent: Programmable privilege control for LLM agents. arXiv:2504.11703.
[26] Kumar, A. et al. (2026). CapChain: A capability-token access control architecture with verifiable provenance for multi-agent LLM systems. Proceedings of IEEE S&P 2026.
[27] Park, S. et al. (2025). AgentSHAP: Interpreting LLM agent tool importance with Monte Carlo Shapley value estimation. Proceedings of NeurIPS 2025.

AI USE STATEMENT

The code implementation described in this paper was written by the authors. The experimental scripts were authored by the authors and executed in a Python 3.13 environment. No AI writing tools were used to draft, revise, or summarise the text of this paper. All theoretical derivations, experimental designs, and interpretations are the authors' own work.

REPRODUCIBILITY STATEMENT

All experimental results in Sections 5.2 through 5.6 can be reproduced using the code included in the supplementary material. Each experiment script accepts a random seed argument; the seeds used in this paper are documented in the script headers. The implementation dependencies are listed in requirements.txt in the supplementary archive. The 74 Python tests can be run with 'pytest tests/' from the project root. The 5 Go chaincode tests require Go 1.21+ and are run with 'go test ./...' from the chaincode/ directory. The 4 Solidity tests require Node.js 18+ and Hardhat, run with 'npx hardhat test' from the saga_bridge/contracts/ directory.