State-verification in asynchronous blockchain-driven agentic AI systems

Gyan Wickremasinghe, Fannya Ratana Sandjaja, Karen Rafferty, Vishal Sharma

School of Electronics, Electrical Engineering and Computer Science (EEECS)
Queen's University Belfast, Northern Ireland, United Kingdom
Email: {Gwickremasinghe01, Fsandjaja, K.Rafferty, V.Sharma}@qub.ac.uk

***Abstract*—With recent advances in Artificial Intelligence (AI), agentic AI systems, capable of perception, reasoning, action, and learning, have accelerated automation across domains. However, their reliance on data raises significant challenges in terms of explicit trust and traceability, which can be addressed using chain-based distributed ledger technology, such as blockchain. While blockchain offers immutability and verifiable state, most existing work focuses on synchronous or partially synchronous models. In contrast, real-world applications utilising agentic AI would benefit from asynchronous blockchains due to their robustness and efficiency. This paper investigates the integration of agentic AI with asynchronous blockchains, where temporal heterogeneity, characterised by previous state observations, is apparent. Despite the advantages of allowing interoperable applications for agentic AI in asynchronous settings, agents querying different nodes may observe divergent states due to network latency, resulting in inconsistent decisions. As a resolution, a timed automaton is combined with an existing Byzantine Fault Tolerance (BFT)-governed gossip protocol and a Zero-Knowledge Proof (ZKP) to ensure consensus safety while allowing controlled heterogeneity and maintaining correctness. Inspired by ensemble learning, the proposed approach treats disagreement not as noise but as a signal for change detection. Experimental results across distributed Virtual Machine (VM) environments demonstrate that BFT compliance with controlled timing significantly reduces ensemble error and improves convergence to ground truth.**

***Index Terms*—Asynchronous Blockchain, State-Verification, Smart Contract, Agentic AI, Fault-Tolerance**

I. Introduction

Distributed Ledger Technology (DLT) follows a secure decentralised architecture which drives properties such as transparency, tamper-proof, and immutability [1]. Blockchain, a chain-based DLT, stores data in the form of transactions secured by cryptographic techniques, but suffers from throughput and latency bottlenecks [1], [2]. Optimisation methods such as sharding, data pruning, and improvements to consensus protocols and layer two solutions have been proposed to address these issues [2]. A key enabler of broader adoption is the smart contract, which is a self-executing program that runs on the blockchain if certain conditions or criteria are satisfied, without the need for verification from a centralised intermediary [1].

Smart contracts have been widely adopted across blockchain platforms such as Solana, Ethereum, and Hyperledger Fabric, enabling applications in supply chains, healthcare, and decentralised finance [1], [3]. However, most deployments rely on synchronous and partially synchronous protocols, which depend on timing assumptions between processes [3]. Asynchronous protocols remove these constraints [4]. However, the Fischer, Lynch, and Paterson (FLP) impossibility shows that no deterministic protocol can maintain both liveness and safety in a fully asynchronous setting [5]. Practical solutions, such as HoneyBadgerBFT's use of the randomised common coin technique [5], have led to further practical asynchronous protocol implementations, including [4], [6], which improve transaction throughput and reduce latency. Consequently, there has been a limited emphasis on integrating smart contracts within asynchronous protocols. A key challenge lies in the querying of a smart contract state from a node, as asynchronous protocols lack guaranteed timing bounds and nodes may not share a consistent view of the blockchain state at any given time.

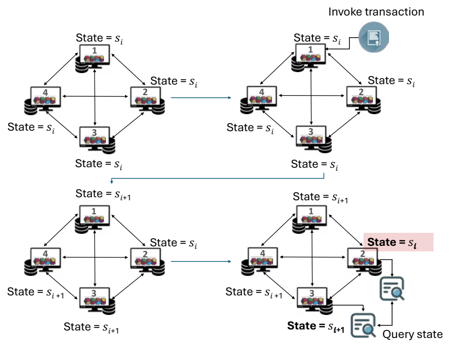


Fig. 1: An exemplary illustration of state variations in an asynchronous blockchain.

In this direction, some initial experimentation, as seen in Fig. 1, demonstrates a proof of the problem. Here, all nodes start at their initial state, and node 1 receives a transaction which invokes a smart contract, resulting in a state change from $s_{1,i}$ to $s_{1,i+1}$. Due to the asynchronous nature, the states of nodes 3 and 4 update to $s_{3,i+1}$ and $s_{4,i+1}$, respectively, while node 2 remains at $s_{2,i}$. An application querying the blockchain from node 2 and node 3 will receive data from different states, causing a fault (variation in the state). To verify this asynchronous nature, an agentic Artificial Intelligence (AI) system is implemented, where four agents query four different nodes, as shown in Fig. 2, revealing a discrepancy in the states of agents when queried in parallel.

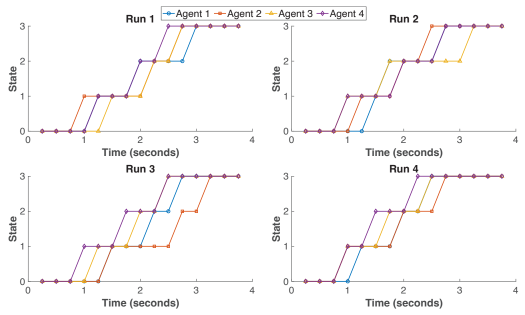


Fig. 2: Difference in states in four agents when querying multiple nodes.

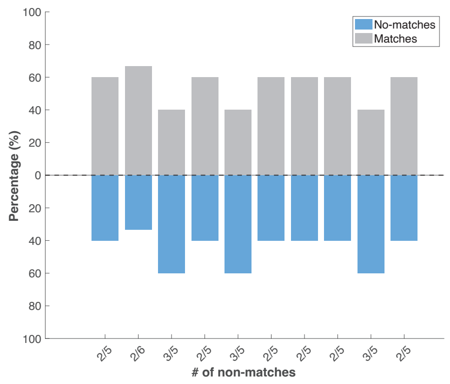


Fig. 3: Percentage mismatch in state information of a blockchain following queries from four agents.

Further experimentation, as seen in Fig. 3, reveals that an average of 37.3% agents were not fault-tolerant when retrieving the most recent state, which further strengthens the need to address this problem. This can be formalised as follows: let L be a distributed ledger run by a set of nodes, $N = \{n_1, n_2, \ldots, n_k\}$. At any time, t, where $t \in \mathbb{R}$, each node, n_k, maintains a local ledger state, $s_{k,i}$ (ith state at time, t). For any two nodes, n_p and $n_q \in N$, each node maintains its set of states as $S_p = \{s_{p,1}, s_{p,2}, ..., s_{p,i}, ...\}$ and $S_q = \{s_{q,1}, s_{q,2}, ..., s_{q,i}, ...\}$. In a fully asynchronous network, there is no bound on message delivery time. Therefore, due to network delays or adversarial behaviour, two nodes can have different local states, i.e., for some time, t_3, $s_{p,i} \neq s_{q,i}$.

With the growing adoption of blockchain in agentic AI for verification, reputation, trust, and integrity, the challenge of consistency remains within an asynchronous protocol [7]. As discussed earlier, orchestrators in a multi-agent system (MAS) (including executing agents in parallel [8]) or the aggregating agents risk making decisions on outdated or inconsistent data, leading to unintended or adversarial outcomes [9]. Therefore, the research question (RQ) remains: *How can applications such as agentic AI maintain the freshness of data when querying multiple blockchain nodes to make decisions?* In resolving this RQ, the main contributions of this work are as follows:

- Including temporal heterogeneity in asynchronous blockchain queries as a source of valid responses to detect state transitions rather than discarding them as faults.
- Including an optimal number of delayed agents, $f = \left\lfloor \frac{\alpha-1}{3} \right\rfloor$, into the framework, keeping in view the safety and diversity of query responses from asynchronous blockchains, where α is the number of agents.
- Implementation of a timed automaton locally to govern state variation within agents, and a global timed automaton within the orchestrator to justify the agents' validity of the final response, where zero-knowledge proof (ZKP) is used to verify agent information.

II. Related Works

Within the context of agentic AI, blockchain has been a growing topic of interest due to the inherent traceability features [7]. Binlashram *et al.* [10] have discussed the use of blockchain to store agent information responsible for detecting failures and errors within system logs, where blockchain stores early warning signals of failures reported by an agent, which is relayed across the network to other agents. However, there is no empirical evaluation of the blockchain's performance in terms of latency impacts (if any) when relaying data across the network.

Dorokhov *et al.* [11] further emphasises blockchain's role in enabling trustless decentralised agents. However, the latency bottleneck within synchronous blockchains can impact agent coordination and access to the most recent state. To mitigate this, Dorokhov *et al.* have proposed a hierarchical multi-blockchain architecture that allows for time-exact transaction processing and verifiable smart contract execution. Additionally, the approach incorporates reinforcement learning-based data sharing to dynamically adapt communication among agents, which improves trust, resilience, and scalable co-ordination in decentralised agents. However, the approach does not consider asynchronous blockchain architecture as a potential solution to the latency and liveness bottlenecks that arise from unexpected delays. Sun *et al.* [12] have discussed trustless and decentralised agents for large-scale agentic AI systems, proposing asynchronous agreement and tokenisation incentives to ensure honest participation. However, concerns over inconsistent state retrieval across nodes persist.

As highlighted above, blockchain can help overcome several limitations within agentic AI systems, such as traceability and smart contract-driven automation. However, there has been limited research on asynchronous blockchains with agentic AI. This work addresses this gap by tackling the inherent problem of querying nodes within an asynchronous protocol for multiple agents.

III. Preliminaries

Ensemble Learning. A core machine learning concept that explores the heterogeneity of results when running different models [13]. The fundamental principle is that an ensemble of diverse models outperforms individual models due to averaging errors and reducing variance, and is calculated using [13],

$$E_{avg} = \frac{1}{M}\sum_{l=1}^{M}\hat{e}_l, \tag{1}$$

where $\hat{e}_l$ is the error of the individual model and M is the total number of models. The ensemble output is based on the mean of all models' predictions,

$$P_m = \frac{1}{M}\sum_{l=1}^{M}\hat{y}_l, \tag{2}$$

where $\hat{y}_l$ is the prediction of the l-th model. The ensemble error is the error of the combined prediction,

$$E_{ens} = (P_m - \beta_{true})^2, \tag{3}$$

which is the mean squared error between P_m and the true value, β_{true}. The reduction, R, is calculated as,

$$R = E_{avg} - E_{ens}. \tag{4}$$

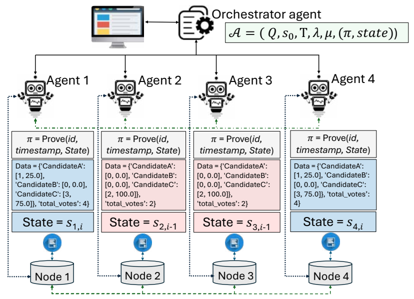
Building on this theory, initial conceptual work has been introduced in surveys on ensemble and perspectives on agentic AI, which include parameters such as diversity and specialisation of agents, collective intelligence, dynamic adaptation, reputation, and trust [14], [15]. This shows that while ensemble learning has been applied within machine learning, further exploration of applications in agentic AI offers new insights, which are increasingly relevant in decentralised scenarios where agent states may differ.

Timed Automata. It is defined as a finite state machine with real-valued clocks. It is used to specify a real value system where constraints limit the behaviour of the automata and is expressed as a tuple [16], $\mathcal{A} = (Q, s_0, T, \lambda, \mu, \sigma)$, where, Q is a finite set of states, $s_0 \in Q$ is the initial state, T is a finite set of clocks, λ is a finite set of actions, $\mu \subseteq Q \times \lambda \times \mathcal{G}(T) \times 2^T \times Q$ is a finite set of edges (or transitions). Here, $\mathcal{G}(T)$ denotes the set of clock constraints (guards) T, and 2^T represents the set of clocks to be reset on the transition. Further, $\sigma : Q \rightarrow \mathcal{G}(T)$ assigns an invariant to each location. A timed automaton verifies the logical flow of the proposed framework to ensure that there are no deadlocks or logical faults when building a solution to the problem of state control and verification.

Zero-Knowledge Proofs. A verification protocol where a receiver can verify the contents shared by a sender without the need to expose private information [17]. Based on the witness, challenge, and response phases, ZKP provides properties of completeness, soundness, and zero-knowledge, which benefits applications such as DLT, ensuring privacy by keeping transaction details, including addresses and amounts, private [17]. Several implementations exist, as discussed in [17], such as pySNARK, Virgo, Halo2, and Limbo, which differ in proof size and the complexity of the prover and verifier. Initially, pySNARK[1] was considered. However, due to ARM architecture compatibility issues *py_ecc*[2] was used to demonstrate the security benefits of ZKP.

IV. Proposed framework

The proposed work resolves state conflicts in MAS by utilising a combination of timed automata and ZKP, and is motivated by ensemble learning. This is built using query validation, utilising asynchronous blockchains to retrieve information. Fig. 4 highlights the proposed framework, where an agent querying the state of the blockchain is susceptible to retrieving different states. While the example in Fig. 4 shows a single agent connecting to one node, it is possible to have multiple agents operate through one node or multiple agents communicating with one node. For example, agents 1 and 4 query nodes 1 and 4, and receive $s_{1,i}$ and $s_{4,i}$, respectively, while agents 2 and 3 query nodes 2 and 3, and receive $s_{2,i-1}$ and $s_{3,i-1}$, respectively. Here, two agents received previous state information. Therefore, a constraint is implemented with the existing gossip protocol [18] to ensure progress only happens when at most only $\frac{1}{3}$ of the total agents have a different state. An exemplary scenario is shown in



Orchestrator agent
$\mathcal{A} = (Q, s_0, \mathrm{T}, \lambda, \mu, (\pi, state))$
Agent 1
Agent 2
Agent 3
Agent 4
π = Prove(*id*, *timestamp, State*)
Data = {'CandidateA': [1, 25.0], 'CandidateB': [0, 0.0], 'CandidateC': [3, 75.0]}, 'total_votes': 4}
Data = {'CandidateA': [0, 0.0], 'CandidateB': [0, 0.0], 'CandidateC': [2, 100.0]}, 'total_votes': 2}
Data = {'CandidateA': [0, 0.0], 'CandidateB': [0, 0.0], 'CandidateC': [2, 100.0]}, 'total_votes': 2}
Data = {'CandidateA': [1, 25.0], 'CandidateB': [0, 0.0], 'CandidateC': [3, 75.0]}, 'total_votes': 4}
State = $s_{1,i}$
State = $s_{2,i\text{-}1}$
State = $s_{3,i\text{-}1}$
State = $s_{4,i}$
Node 1
Node 2
Node 3
Node 4

Fig. 4: Handling queries from multiple nodes to ensure consistency across AI agents.

Fig. 5, where each agent communicated with another agent at random for up to five rounds, based on clock *C2*, which is defined as a part of the timed automata, until at least three agents agree on the same state. While combining *BFT* with *timed automata*, and the concepts from *ensemble learning*, the orchestrator will only proceed after ensuring that

1) agents ensure that BFT is satisfied amongst themselves,
2) agents with previous states are still accepted within the round, providing data heterogeneity via timed automata, and
3) all agents return a valid response within a pre-defined number of rounds.

These conditions are managed by algorithm 1, where the random gossip protocol [18] is used. Within each round, an agent randomly verifies their state with two other agents. In a system of α agents, the number of f agents that do not retrieve the same state can be defined as [19],

$$f = \left\lfloor \frac{\alpha - 1}{3} \right\rfloor. \tag{5}$$

[1]https://github.com/meilof/pysnark, last accessed - 25/09/2025

[2]https://github.com/ethereum/py_ecc, last accessed - 25/09/2025

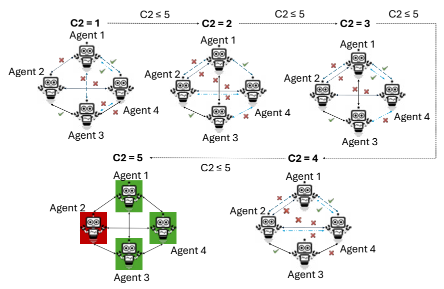


Fig. 5: Random gossip between agents based on timed automata with clock, *C2*.

Algorithm 1 Response comparison using Gossip Protocol

```
Require: Agent ID: $id$, validator: $(ip, port)$, results queue: $result\_queue$
Ensure: Final voting state and consensus status
 1: Initialise $LTA \leftarrow$ LocalTimedAutomaton($id$)
 2: while $LTA$.round_counter $< LTA$.rounds do
 3:     $success, result \leftarrow$ CompleteRound($LTA, node\_ip,$
            $node\_port, result\_queue$)
 4:     if $success \wedge$ result $\neq$ null $\wedge$ result.bft_met then
 5:         return $(result)$
 6: $fallback \leftarrow (InitialValue)$
 7: return
      $(id,\ fallback,$
      $\{C1,\ C2\},$
      round : $LTA$.round_counter, final_state : "FAILED")
```

Here, the minimum quorum size of agreement is given by [19],

$$\delta = 2f + 1 = 2\left\lfloor \frac{\alpha - 1}{3} \right\rfloor + 1, \tag{6}$$

and, f refers to non-malicious agents that have diverged in state, excluding malicious agents whose security impact is beyond the scope of this paper. Since information is disseminated to all nodes in $O(\log |N|)$ rounds with high probability [20], the total number of δ consistent states is observed between 2 and 3 rounds in a setup with four agents. Therefore, setting $\chi = 5$ (rounds) provides a safe upper limit[3].

Algorithm 1 is governed by algorithm 2, where each agent follows its own local timed automaton. Here, the *QueryNode* function receives a response from the blockchain smart contract and is stored within a variable *own_state*. Following this, the agent generates a cryptographic commitment, π, which takes in the parameters, *id, current_state, timestamp*, driven by ZKP to allow verifying the underlying state. Random gossip selects two agents based on the *RandomPeer* function, where the $SendGossip$ function is responsible for invoking the existing gossip protocol [18]. The states are verified between agents by retrieving peer states using the *collectedPeerStates* function, which consists of a list with dictionaries of responses from other agents. Algorithm 3 utilises a timed automaton based on the outputs of the local timed automata from Algorithm 2, which returns a tuple of a boolean indicating if

[3]For the sake of verification and experimentation, rounds that do not satisfy BFT are also considered valid to show the impact on the ensemble error. However, identifying the upper limit is beyond the scope of this work and needs further empirical evaluation or asymptotic analysis.

Algorithm 2 Local Timed Automaton: Consensus Cycle

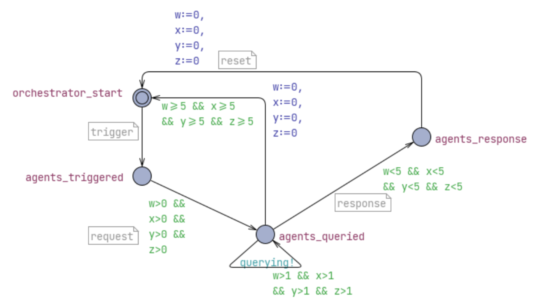
```
Require: Agent ID: $id$, validator: $(ip, port)$, rounds = $\chi$
Ensure: Success flag, result with BFT status
 1: if rounds $> \chi$ then
 2:     return (false, null)
 3: $C1 \leftarrow C1 + 1$                                   ▷ Phase 1: Query blockchain
 4: $own\_state \leftarrow$ QueryNode($current\_state, ip, port$)
 5: if $own\_state =$ null then
 6:     return (false, null)
 7: $\pi \leftarrow$ ProveValidQuery($id, current\_state, timestamp$)
 8: $C2 \leftarrow C2 + 1$                                   ▷ Phase 2: Gossip to peers
 9: $success\_count \leftarrow 0$
10: for each round $\chi$ do
11:     $peer1, peer2 \leftarrow$ RandomPeer()
12:     SendGossip($peer1, current\_state, \pi$)
13:     SendGossip($peer_2, current\_state, \pi$)
14:     if gossip succeeds then
15:         $success\_count \leftarrow success\_count + 1$
16: $peer\_states \leftarrow$ CollectedPeerStates()           ▷ Phase 3: Check BFT
17: $valid\_states \leftarrow \{S \mid (S, \pi) \in peer\_data \wedge$ Verify$(\pi)\}$
18: $all\_states \leftarrow \{current\_state\} \cup valid\_states$
19: if $|all\_states| < 2$ then
20:     return (false, null)
21: $list\_states \leftarrow \{$dict_to_list$(S) \mid S \in all\_states\}$
22: $(majority\_state, count) \leftarrow$ counter.most_common($list\_states$)
23: $bft\_met \leftarrow (count \geq 3)$
24: $result \leftarrow (id, current\_state, bft\_met, C1, C2)$
25: return (true, $result$)
```

Algorithm 3 Global Timed Automaton: Orchestrator

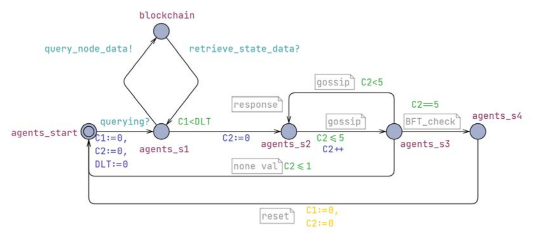
```
Require: rounds = $\chi$
 1: while not converged and rounds $< \chi$ do
 2:     rounds $\leftarrow$ rounds + 1
 3:     Reset all agent clocks $\{w, x, y, z\}$
 4:     Broadcast "vote" command to all agents
 5:     while waiting for $responses$ do
 6:         Receive $(agent\_id, result)$
 7:         if Verify($result.\pi$) then
 8:             $responses \leftarrow responses \cup \{result\}$
 9:         else
10:             Discard invalid proof
11:         Increment clocks: $w, x, y, z \leftarrow w + 1,\ x + 1,\ y + 1,\ z + 1$
12:         if $w = x = y = z =$ rounds then
13:             break
14:     if $((responses.bft\_met) \geq 3)$ then
15:         return (true, $responses$)
16: return (false, $responses$)
```

states matched, followed by the result, which is a dictionary containing *id, current_state, bft_met, C*1*, and C*2. Initially, the orchestrator requests all agents to query the blockchain, which is executed in parallel. A function *Receive* retrieves the agent responses, and the *Verify* function ensures agent data is verified based on ZKP. Finally, the orchestrator will check if the condition for BFT is met, defined by a counter, *bft_met*, which determines whether to request another blockchain query or make a decision based on the received responses.

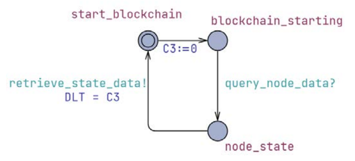
The proposed framework's workflow is formally verified using UPPAAL [21], which helps to validate the behaviour of the timed automata, as shown in Fig. 6. Here, Fig. 6a consists of the timed automata responsible for ensuring the execution of all agents, each with an individual clock, *w, x, y, z*. All four agents are triggered in parallel, each interacting with a separate blockchain node. As illustrated in Fig. 6b, each



(a) Orchestrator's workflow.



(b) Agent's workflow.



(c) Blockchain's workflow.

Fig. 6: Verification of system components with timed-automata.

agent is executed through a local timed automaton. There are three clocks: *C1* is the default clock of the agent for querying the blockchain, *C2* is for the gossip round, and the third clock, *DLT*, is for the blockchain. Whenever the blockchain is queried through *query_node_data*, the *DLT* clock is updated. Fig. 6c is the timed automaton for the blockchain. When the agent queries the node, it returns the state of the blockchain, where *C3* is the clock that runs within the blockchain. The verification helps ensure that there are no deadlock processes within the entire system, allowing the orchestrator to continue executing.

V. Experimental Results

The proposed framework, initiated under asynchronous conditions, consisted of four nodes running the Dumbo-NG protocol [6] with a batch size of 500 transactions, each comprising 20 slots. Transactions were deployed across the network, which included smart contract triggering transactions that utilised a virtual machine (VM) for off-chain smart contract execution. An orchestrator with four independently-deployed agents to communicate with the blockchain nodes. The approach was developed in Python 3.7 and deployed on Azure using Ubuntu Server 22.04 LTS (ARM64 Gen2) with a standard D2ps v5 instance (2 vCPUs, 8 GB RAM). All experiments were conducted using the same configuration to ensure comparable conditions, and VMs were deployed across East US, Australia East, Canada Central, North Europe, West US, Sweden Central, Japan East, and Central India. The orchestrator ran for thirty seconds, starting from the initial state to the fifth state. The state comprises a voting system consisting of three candidates, which gets updates based on the invoked transaction. The orchestrator's role was to trigger agents to fetch the state of the nodes. Each triggered agent queried the state of the smart contract every two milliseconds. This timestamp helps to understand the asynchronous nature of state updates, as four agents querying within the same time window may observe different states depending on when updates are committed. States were then shared with other agents to decide and verify, before the final result was shared with the orchestrator.

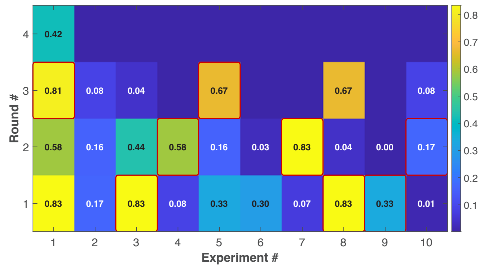


Fig. 7: Average individual error of agents per round across multiple systems.

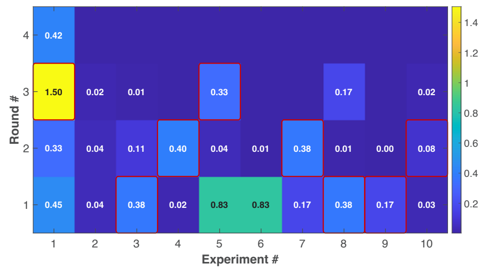


Fig. 8: Ensemble error of agents per round across multiple systems.

Since nodes within the blockchain network behave asynchronously, this can influence the baseline due to the variation in state among agents. This means that running experiments with the proposed approach can yield different baselines across different systems, even with similar settings. Despite this variation, the difference between rounds that adhere to BFT and those that do not will be evident. As shown in Fig. 7, the average individual error increases significantly in each experiment when the timed automata conditions were not met. For example, in Experiment 2, where all rounds satisfy the timed automata, the average individual error remains consistently low. In contrast, Experiment 5 shows a sharp rise in the mean square error from 0.33 to 0.67, which demonstrates the impact of rounds violating the timed automata, indicating greater deviation from the ground truth. Fig. 8 displays the ensemble error, where a similar trend emerges - higher ensemble error

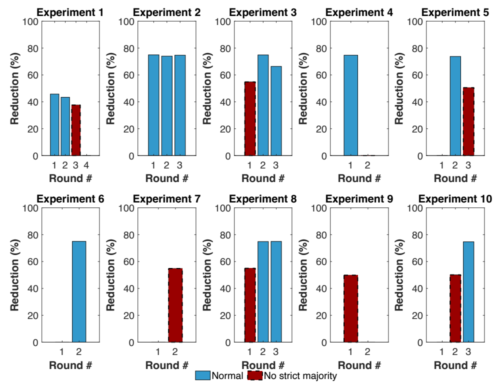


Fig. 9: Reduction gain in four agents vs. number of rounds.

occurs when $f \geq |N|/3$, reflecting that agents deviate from the ground truth due to insufficient majority agreement. For example, in round 3 of Experiment 1, the ensemble error increases from 0.33 to 1.5, where the constraints for the timed automata are not compliant, suggesting a greater deviation from the truth value due to the increased average individual error. Finally, Fig. 9 shows the relative reduction gain using $R_g = (1 - \frac{E_{\text{avg}}}{E_{\text{ens}}}) \times 100\%$. Based on the experimental results, reduction remains high across all rounds where constraints for the timed automata are not compliant. For example, in Experiments 1 and 5, the reduction percentage on average decreased by 15.7%, indicating a declining impact on improvement across the agents' responses. Similarly, Experiments 3, 8 and 10 show an average improvement in reduction gain of 14.96% across rounds that satisfy timed automata conditions.

VI. Conclusion

This paper explored the use of asynchronous blockchains with smart contract integration in agentic Artificial Intelligence (AI) systems. The proposed framework helps solve the problem of state verification among agents, combining timed automata for freshness and validation with randomised gossip to minimise communication overhead while preserving verifiability through zero-knowledge proof (ZKP). Results show that a timed automata approach with ZKP and ensemble learning achieves stable performance, with a significant improvement in reduction gain. This work demonstrates that temporal heterogeneity can be enhanced, opening new pathways for resilient agentic AI systems on asynchronous blockchains.

Acknowledgements

This research is supported by the Advanced Research Engineering Centre (ARC) project, funded by PwC and Invest NI. The views expressed are those of the authors and do not necessarily represent the project/funding agency.

References

[1] X. Guo, Y. Zuo, and D. Li, "When auditing meets blockchain: A study on applying blockchain smart contracts in auditing," *International Journal of Accounting Information Systems*, vol. 56, p. 100730, 2025.

[2] H. Y. Wu, X. Yang, C. Yue, H.-Y. Paik, and S. S. Kanhere, "Chain or dag? underlying data structures, architectures, topologies and consensus in distributed ledger technology: A review, taxonomy and research issues," *Journal of Systems Architecture*, vol. 131, p. 102720, 2022.

[3] S. D. Angelis, F. Lombardi, G. Zanfino, L. Aniello, and V. S. and, "Security and dependability analysis of blockchain systems in partially synchronous networks with byzantine faults," *International Journal of Parallel, Emergent and Distributed Systems*, pp. 1–21, 2023.

[4] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, (New York, NY, USA), p. 803–818, Association for Computing Machinery, 2020.

[5] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), p. 31–42, Association for Computing Machinery, 2016.

[6] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, (New York, NY, USA), p. 1187–1201, Association for Computing Machinery, 2022.

[7] Z. Guo, Y. Zhou, C. Wang, L. You, M. Bian, and W. Zhang, "Betaweb: Towards a blockchain-enabled trustworthy agentic web," *arXiv preprint arXiv:2508.13787*, 2025.

[8] K.-T. Tran, D. Dao, M.-D. Nguyen, Q.-V. Pham, B. O'Sullivan, and H. D. Nguyen, "Multi-agent collaboration mechanisms: A survey of llms," *arXiv preprint arXiv:2501.06322*, 2025.

[9] K. Huang, V. S. Narajala, J. Yeoh, J. Ross, R. Raskar, Y. Harkati, J. Huang, I. Habler, and C. Hughes, "A novel zero-trust identity framework for agentic AI: Decentralized authentication and fine-grained access control," *arXiv preprint arXiv:2505.19301*, 2025.

[10] A. Binlashram, H. Bouricha, L. Hsairi, and H. Al Ahmadi, "A new multi-agents system based on blockchain for prediction anomaly from system logs," in *Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services*, iiWAS '20, (New York, NY, USA), p. 467–471, Association for Computing Machinery, 2021.

[11] I. Dorokhov, J. Ruponen, R. Schutski, and A. Nechesov, "Time-exact multi-blockchain architectures for trustworthy multi-agent systems," in *First Conference of Mathematics of AI*, pp. 1–19, 2025.

[12] R. Sun, Z. Wang, J. Sun, and R. Ranjan, "Vision: How to fully unleash the productivity of agentic AI? decentralized agent swarm network," in *ICML 2025 Workshop on Collaborative and Federated Agentic Workflows*, pp. 1–12, 2025.

[13] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Advances in Neural Information Processing Systems* (G. Tesauro, D. Touretzky, and T. Leen, eds.), vol. 7, MIT Press, 1994.

[14] E. Miehling, K. N. Ramamurthy, K. R. Varshney, M. Riemer, D. Bouneffouf, J. T. Richards, A. Dhurandhar, E. M. Daly, M. Hind, P. Sattigeri, *et al.*, "Agentic AI needs a systems theory," *arXiv preprint arXiv:2503.00237*, 2025.

[15] R. Sapkota, K. I. Roumeliotis, and M. Karkee, "AI agents vs. agentic AI: A conceptual taxonomy, applications and challenges," *Information Fusion*, vol. 126, p. 103599, Feb. 2026.

[16] R. Alur, "Timed automata," in *Computer Aided Verification* (N. Halbwachs and D. Peled, eds.), (Berlin, Heidelberg), pp. 8–22, Springer Berlin Heidelberg, 1999.

[17] E. Morais, T. Koens, C. van Wijk, and A. Koren, "A survey on zero knowledge range proofs and applications," *SN Applied Sciences*, vol. 1, no. 8, p. 946, 2019.

[18] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[19] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, p. 398–461, Nov. 2002.

[20] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, "Randomized rumor spreading," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 565–574, 2000.

[21] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL - a tool suite for automatic verification of real-time systems," in *Hybrid Systems III* (R. Alur, T. A. Henzinger, and E. D. Sontag, eds.), (Berlin, Heidelberg), pp. 232–243, Springer Berlin Heidelberg, 1996.