A Multi-Agent AI Framework for Automated Research Data Analysis with Blockchain-Based Audit Trail

Tajammul Pasha
M.Tech Student, Department of Computer Science and Engineering, Ghousia College of Engineering, Ramanagaram, Karnataka-562159, India, Affiliated VTU Belagavi.

Dr. Dilshad Begum
Professor and Head, Department of Computer Science and Engineering, Ghousia College of Engineering, Ramanagaram, Karnataka-562159, India

Dr. Moinuddin S K
Assistant Professor, Department of Robotics and Artificial Intelligence, Ghousia College of Engineering, Ramanagaram, Karnataka-562159, India

Abstract
Autonomous multi-agent systems built on large language models (LLMs) can carry out entire analytical workflows with little human direction, yet they provide no built-in way to prove what each agent actually did: intermediate reasoning is ephemeral and outputs can be altered without leaving any trace. In this work, a nine-agent autonomous research pipeline implemented using LangGraph and LangChain is presented, which accepts a tabular CSV dataset and a stated research objective and returns statistical analysis, machine-learning models, visualisations and a complete HTML research report. Accountability is attached through a non-invasive instrumentation layer (node_hook.py) that intercepts every agent output, serialises it to a canonical form, computes its keccak256 digest and commits the digest to an append-only smart contract, AgentAuditTrail, on an EVM-compatible blockchain network, producing an immutable and independently verifiable audit trail. Evaluated on five real-world CSV datasets, the pipeline completed every task end to end, achieved an average report-quality rating of 4.12 out of 5.0 from independent reviewers, incurred a blockchain audit-trail overhead of 6.2% of total runtime, and detected 100% of injected modifications across 50 controlled tamper experiments.

Novelty in the Research: Existing multi-agent frameworks such as AutoGen, CrewAI and LangGraph provide no tamper-evident record of agent execution, and prior blockchain-provenance systems secure data at rest or in transit rather than the step-by-step behaviour of AI agents. The closest related architecture records agent actions on a private, permissioned chain and requires invasive changes to agent code. In contrast, the present work is, to the best of the authors' knowledge, the first to combine a multi-agent research pipeline with a publicly verifiable blockchain audit trail through a non-invasive hook that requires only two lines of integration code and no modification of any agent, and to quantify the cost of that guarantee (6.2% runtime overhead, about 67,200 gas per action) together with a 100% tamper-detection rate.

Keywords: Multi-agent AI, LangGraph, Blockchain, Audit Trail, Data Provenance, Large Language Models, Smart Contracts, Solidity

DOI: https://doi.org/10.67317/IJMSRT26JUL026

1. Introduction

The maturation of large language models into general-purpose reasoning engines, built upon the transformer architecture [1], has reshaped the way computational research is conducted. Contemporary LLMs serve as the cognitive core of autonomous agents that decompose an objective into intermediate reasoning steps, interleave reasoning with the invocation of external tools, and revise their own conclusions through self-reflection [2]. When several such agents are composed, each assuming a specialised role, the result is a multi-agent system able to execute multi-step analytical workflows with little human supervision. Frameworks such as AutoGen [3], which structures agent interaction as conversation, CrewAI, which organises agents into role-bearing teams, and LangGraph, which represents a workflow as a stateful directed graph, orchestrate this collaboration, and research systems such as MetaGPT [4] demonstrate that assigning differentiated roles measurably improves the quality of the collective output. Such pipelines are increasingly used to generate hypotheses, write and execute analysis code, produce visualisations and draft structured reports across domains as varied as healthcare, finance and the natural sciences [2].

This leap in capability introduces a class of problems that did not previously exist. When a multi-agent system autonomously produces a research artefact, the provenance of that artefact - precisely which agent performed which action, in what order and on what inputs - is neither recorded nor independently verifiable once the run has completed. The intermediate reasoning is ephemeral, the outputs are mutable, and the only trace of how a conclusion was reached is whatever the system happens to retain. Surveys of LLM-based agents repeatedly identify trust, accountability and verifiability - rather than raw capability - as the binding constraints on real-world deployment [2, 5].

Conventional mechanisms for recording system behaviour - application log files, relational audit tables and centralised monitoring platforms - share a structural weakness: they are mutable and are administered by the very party whose behaviour they purport to document. Any actor with write access to the logging substrate can silently alter, reorder or delete records without leaving an independent trace [10]. Classical secure-logging schemes render logs tamper-evident through cryptographic chaining [10, 11], but they still depend on a trusted administrator and offer no means for an external reviewer to verify integrity without privileged access. The need for verifiable provenance is also increasingly a regulatory expectation: the European Union's Artificial Intelligence Act [9] imposes record-keeping and traceability obligations on high-risk AI systems. In parallel, the research community has documented a reproducibility crisis in artificial intelligence, in which a large fraction of reported results cannot be independently reproduced [6], and work on algorithmic accountability and model auditing [7, 8] argues that trustworthy AI demands an unbroken, inspectable evidentiary chain from inputs to outputs.

Blockchain technology offers a structural answer to the mutability problem. A blockchain is an append-only, cryptographically linked, replicated ledger whose contents cannot be altered retroactively without detection [12], and its programmable smart-contract variant records application logic and state transitions on-chain [13]. These integrity properties have been exploited for data provenance in cloud environments [15], decentralised provenance stores [16], and medical-record audit trails [17], and blockchain-backed logging systems replace the trusted administrator with public consensus [18, 19]. The recurring architectural pattern in this literature is hash-and-store: only a compact cryptographic commitment is recorded on-chain while the full payload is retained off-chain [15, 16]. A blockchain earns its complexity only when parties who do not fully trust one another must agree on a shared, unalterable state and no trusted intermediary is acceptable [14] - precisely the position of an external reviewer who must verify an operator's claims without trusting that operator.

Despite this maturity, the application of on-chain provenance to the outputs of autonomous LLM agents remains largely unexplored. The most directly related prior work couples a LangChain-based multi-agent system to a permissioned blockchain that enforces policy and records actions [25]; it relies, however, on private institutional infrastructure rather than an openly verifiable network, and requires invasive integration with the agent codebase.

Accordingly, the research question investigated here is whether a blockchain audit layer can be attached to an autonomous multi-agent research pipeline in a non-invasive manner, such that every agent action becomes independently and permanently verifiable, while the resulting performance overhead is kept within bounds acceptable for research use. Four objectives follow: (1) to design a

DOI: https://doi.org/10.67317/IJMSRT26JUL026

nine-agent autonomous research pipeline for the end-to-end analysis of tabular CSV datasets, producing statistical analyses, machine-learning models, visualisations and a self-contained HTML research report; (2) to integrate a blockchain-based audit-trail layer into this pipeline without modifying the logic of any individual agent; (3) to evaluate the pipeline across five datasets of differing size and domain, establishing that the audit layer's overhead remains below a ten-percent budget; and (4) to demonstrate the tamper-resistance of the audit trail through controlled modification experiments. The remainder of this paper is organised as follows: Section 2 presents the materials and methodology, Section 3 reports and discusses the results, and Section 4 concludes.

2. Materials and Methodology

2.1 Overall Architecture

The system is organised into three decoupled layers that execute sequentially during a pipeline run. Layer 1, the multi-agent research engine, is a LangGraph StateGraph composed of nine specialised agents operating over a typed Pydantic state object, with conditional routing and loop guards; it accepts a CSV dataset and a research objective and produces a self-contained HTML report. Layer 2, the blockchain audit trail, is a non-invasive hook (node_hook.py) that intercepts the execution of each agent node, serialises the node's output to canonical JSON, computes its keccak256 digest [23] and records that digest on-chain through the AgentAuditTrail smart contract. Layer 3 emits three artefacts: report.html, session_audit.json (the complete local audit trail with payloads, hashes and transaction references) and the set of publicly queryable on-chain records. Two cross-cutting principles govern the design: non-invasiveness, so that the act of auditing cannot itself alter the behaviour being audited, and graceful degradation, so that if the blockchain is unreachable the research engine continues to function unimpaired.

2.2 Multi-Agent Research Engine

Each agent embodies the single-responsibility principle: it is responsible for exactly one well-defined transformation of the shared state. An agent's behaviour is specified declaratively through a prompt file that defines its role and a YAML configuration that parameterises it, so that its conduct can be adjusted without editing code. At runtime an agent receives the typed state, performs its function and returns a dictionary of field updates that LangGraph merges into the shared state under Pydantic validation. Because every agent communicates exclusively through its typed return value, that value is a complete and faithful object to hash. The nine agents and their primary responsibilities are summarised in Table 1.

Table 1: Nine-Agent Pipeline - Roles and Primary Outputs

Agent	Role	Primary Output
Hypothesis	**Generates hypotheses from metadata**	**hypothesis field**
Process	**Plans analysis tasks**	**todo_list**
Search	**Retrieves external knowledge**	**search_artifacts**
Code	**Writes and executes analysis code**	**code_artifacts**
Visualisation	**Generates visualisation scripts**	**data_viz_artifacts**
Report	**Drafts the HTML report**	**report_artifacts**
QualityReview	**Evaluates report quality**	**quality_feedback**
Refiner	**Applies quality feedback**	**report_artifacts**
Note (NoteTaker)	**Truncates message history**	**messages (trimmed)**

Control flow is governed by routing functions rather than a fixed sequence. After every Process-agent execution, a router inspects the current task list and dispatches to the appropriate specialist agent; after the report is drafted, a quality-review router routes to the Refiner agent when revision is warranted. Two orthogonal loop guards guarantee termination: a global step counter terminates the graph when it exceeds fifty node executions, and a revision counter bounds the revision loop at two iterations. To keep long runs within the model's context window, the NoteTaker agent applies a sliding-window truncation to the message history, retaining the two earliest and six most recent messages while

DOI: https://doi.org/10.67317/IJMSRT26JUL026

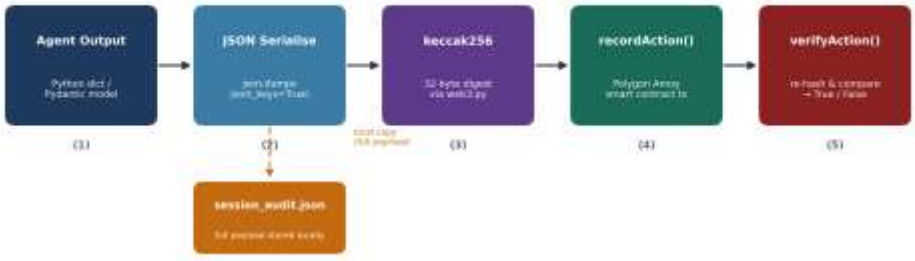
exempting the task state from truncation. The pipeline is implemented in Python 3.12 on LangGraph 0.2 over LangChain 0.3, with LLM inference served through the Anthropic Claude gateway, and the audit contract is written in Solidity 0.8.20 and accessed through web3.py 6.

2.3 Blockchain Audit Trail Layer

The audit layer satisfies four design requirements: it is tamper-evident (storage is append-only, with no update or delete operation exposed at any level of the interface), publicly verifiable (any third party can verify an agent-output hash against the on-chain record without trusting the party that recorded it), lightweight (only a 32-byte keccak256 digest is stored on-chain), and non-invasive (no change to existing agent code is required). The adversary of interest is an insider - the operator of the pipeline, or any party with access to its local storage - who, after a run has completed, wishes to alter a recorded output and have the altered version pass as authentic. Under this model the security goal is tamper-evidence rather than tamper-prevention: the system does not stop an adversary from editing a local payload, but it guarantees that any such edit can be detected by re-hashing the payload and comparing it against the immutable on-chain commitment.

For every agent action the payload is serialised to canonical JSON with a deterministic encoder (sorted keys, normalised container and floating-point types), so that semantically identical payloads always yield byte-identical strings and therefore identical digests. The keccak256 hash of this string is computed and only the resulting 32-byte digest, together with the session identifier, the agent name and an action type, is written on-chain; the full payload is retained locally in session_audit.json. Verification is symmetric and trustless: any party in possession of a payload re-serialises and re-hashes it and calls the contract's verifyAction function, which returns true only if the recomputed digest matches the stored commitment. Because keccak256 exhibits the avalanche property [23], even a single-character alteration of the payload produces an unrelated digest. This hash-and-store pattern, inherited from the blockchain-provenance literature [15, 16], is illustrated in Fig. 1.

Fig. 1: Hash-and-Store Blockchain Recording Pattern



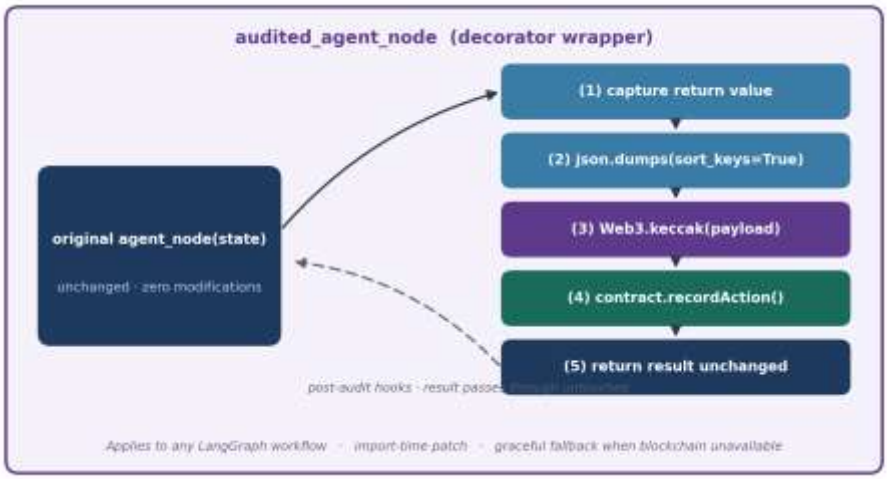
The AgentAuditTrail smart contract defines a single AgentAction record comprising a session identifier, the agent name and action type, the payload digest, a block timestamp and the address of the recording account. Records are appended to a public actions array, and a mapping from session identifier to action indices allows all actions belonging to a given run to be retrieved efficiently. The contract exposes four functions: recordAction appends a new record and emits an indexed event; verifyAction compares a supplied digest against a stored record; getSessionActions returns all action indices for a session; and totalActions returns the global count. The contract's security rests as much on what it omits as on what it provides: it defines no function capable of updating or deleting a stored record, so immutability is enforced at the level of the contract's interface. This minimalism also shrinks the attack surface that smart-contract security analyses warn against [21, 22]. The contract was deployed on a local Hardhat node, an Ethereum-compatible EVM test network chosen for its low latency and zero economic cost; migration to a public production network such as Polygon is identified as future work, and the per-action cost of such a migration is quantified in Section 3.2.

DOI: https://doi.org/10.67317/IJMSRT26JUL026

2.4 Non-Invasive Hook: node_hook.py

The non-invasive requirement is realised through the decorator pattern, shown in Fig. 2. A wrapper function takes an existing agent-node function and returns a replacement with an identical signature: on each invocation it calls the original function, captures its return value, serialises that value to canonical JSON, computes the keccak256 digest and submits a recordAction transaction, and then returns the original value unchanged. A companion function iterates over the node module and replaces every agent node with its audited counterpart, after which the workflow module is reloaded so that the compiled graph references the wrapped functions. Two lines of integration code thus render an entire existing workflow auditable. If the environment variables that supply the contract address and the signing key are absent, the hook degrades gracefully to a local-only mode, and any failure during recording - a dropped connection, an exhausted nonce, a rejected transaction - is caught and logged but never allowed to propagate, so an audit fault can never abort a research run.

Fig. 2: Decorator Architecture of the Non-Invasive Hook



2.5 Experimental Setup

All experiments were performed on Apple M-series hardware with 16 GB of RAM, using the Anthropic Claude gateway as the LLM provider. Five CSV datasets spanning a range of sizes (100 to 1,200 rows) and domains were selected to probe the pipeline's generalisability across analysis types, as listed in Table 2. Every run used the full-analysis depth with automatic hypothesis selection. Report quality was assessed by two independent evaluators against a four-dimension rubric - hypothesis relevance, code correctness, chart clarity and report coherence - each scored on a five-point scale, and inter-rater reliability was quantified with Cohen's kappa [20]. To isolate the cost attributable to auditing, each dataset was additionally run with the audit hook disabled, providing a no-audit baseline against which both runtime overhead and output equivalence were assessed. The integrity guarantee was validated directly through 50 controlled tamper experiments: in each, a single character of a recorded agent output was altered in session_audit.json, the modified payload was re-hashed with keccak256, and the result was checked against the corresponding on-chain record through verifyAction.

Table 2: Evaluation Datasets

Dataset	Rows	Domain	Analysis Goal
people-100.csv	100	Demographics	Exploratory Data Analysis
iris.csv	150	Biology	Species Classification
finance-monthly.csv	365	Finance	Time-Series Forecasting
customer-data.csv	500	Retail	Customer Clustering
sales-2024.csv	1,200	Sales	Regression Analysis

DOI: https://doi.org/10.67317/IJMSRT26JUL026

3. Results and Discussion

3.1 Pipeline Performance

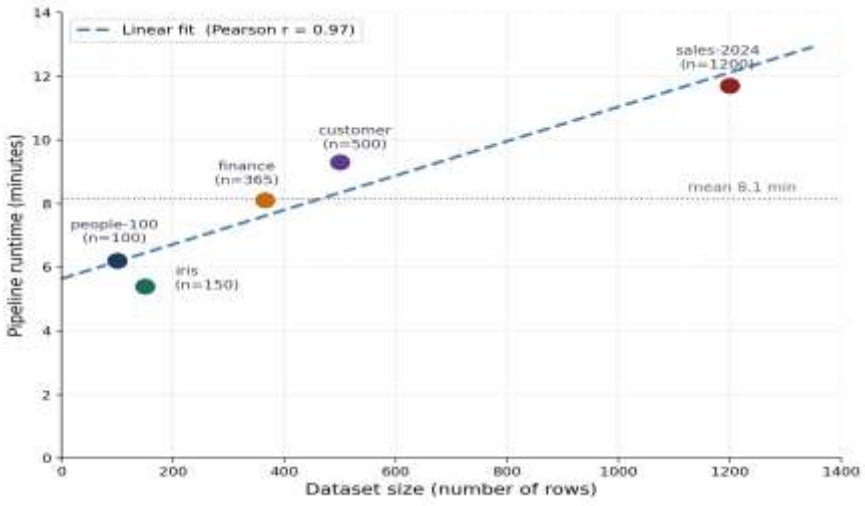
Across all five datasets the pipeline achieved a 100% task-completion rate: every run produced a complete HTML report without manual intervention. No run reached either the step-count ceiling of fifty or the revision-count ceiling of two, indicating that termination was reached through normal completion rather than through a guard firing. Table 3 reports completion status, runtime and mean quality per dataset. The mean quality across all datasets and rubric dimensions was 4.12 out of 5.0 (standard deviation 0.24). The iris classification task scored highest at 4.60, consistent with its well-structured and extensively studied nature, while the largest dataset, sales-2024.csv, scored lowest at 3.90, suggesting that report quality degrades modestly as dataset size and analytical complexity grow. Inter-rater agreement was strong, with Cohen's kappa of 0.81 [20], lending confidence that the quality scores reflect genuine differences rather than rater idiosyncrasy.

Table 3: Pipeline Completion, Runtime and Quality Scores

Dataset	Completion	Runtime (min)	Avg Quality (/5)
people-100.csv	**100%**	**6.2**	**4.08**
iris.csv	**100%**	**5.4**	**4.60**
finance-monthly.csv	**100%**	**8.1**	**4.00**
customer-data.csv	**100%**	**9.3**	**3.98**
sales-2024.csv	**100%**	**11.7**	**3.90**
Mean	**100%**	**8.14**	**4.12**

Disaggregating the mean across the four rubric dimensions is informative. Code correctness and chart clarity were consistently the strongest dimensions, reflecting the maturity of the underlying scientific-Python tooling and the relatively constrained space of sensible visualisations for tabular data. Report coherence was strong for the smaller, well-structured datasets but declined for the largest, where the volume of intermediate results strained the sliding-window context management. Hypothesis relevance showed the widest spread, being excellent on the well-characterised iris case and weakest where a dataset's semantics were ambiguous. This pattern is consistent with the failure modes catalogued in the agent literature [2]: the pipeline is most dependable when the task is well posed and degrades gracefully, rather than catastrophically, as ambiguity grows. The mean runtime was 8.14 minutes (standard deviation 2.34) and, as shown in Fig. 3, runtime correlated strongly and almost linearly with dataset size (Pearson r = 0.97), indicating predictable scaling behaviour. The dominant cost is LLM-API latency rather than local computation.

Fig. 3: Pipeline Runtime versus Dataset Size



DOI: https://doi.org/10.67317/IJMSRT26JUL026

3.2 Blockchain Audit Trail Results

Across the five runs, 73 agent actions were recorded on-chain, a mean of 14.6 per session (standard deviation 2.1). The mean block-confirmation time was 2.1 seconds (standard deviation 0.4), matching the network's nominal two-second block interval, and no transaction failed to confirm. Each recordAction call consumed approximately 67,200 gas (standard deviation 1,840), a figure that reflects the deliberately minimal contract logic and the small, fixed-size payload written per action. On the test network the monetary cost is zero; extrapolated to Polygon mainnet at a representative gas price of 30 gwei, the equivalent cost would be on the order of 0.002 USD per action, which is negligible for research use and modest even at production volumes.

Table 4 consolidates the audit-trail performance metrics. The mean total blockchain overhead was 30.7 seconds per session (standard deviation 5.2), equivalent to 6.2% of total pipeline runtime (standard deviation 1.1%). This sits comfortably below the 10% budget set as a design target, confirming that blockchain auditing can be added to a research pipeline at a cost that does not materially affect the user experience. In the tamper-detection experiment, all 50 controlled modifications returned false from verifyAction, yielding a 100% tamper-detection rate. This outcome is the empirical manifestation of the avalanche property of the underlying hash function [23]: a one-character edit produces a digest unrelated to the original, so no realistic modification can escape detection.

Table 4: Blockchain Audit Trail Performance Metrics

Metric	Mean	Std Dev
Actions per session	14.6	2.1
Tx confirmation time (s)	2.1	0.4
Gas per recordAction	67,200	1,840
Total overhead (s)	30.7	5.2
Overhead (% of runtime)	6.2%	1.1%

3.3 Discussion

The decorator architecture met its primary design goal: not one of the nine agent implementations required modification to become auditable, because the hook operates at the workflow level and therefore captures all current and future agents automatically. In a controlled comparison, running each dataset with and without the hook produced byte-identical HTML reports, confirming empirically that the instrumentation is transparent to the pipeline it observes.

The measured 6.2% overhead is dominated by the roughly two-second confirmation latency of each per-action transaction. This cost is a direct consequence of the design choice to record every action individually, which maximises audit granularity. It could be reduced to a single transaction per run by aggregating all action digests into a Merkle root [24] and committing only that root on-chain, lowering the transaction count from O(n) to O(1) at the expense of per-action verifiability. It is also instructive to situate the measured overhead against the alternative it replaces: a conventional centralised audit store imposes negligible write latency but provides none of the guarantees enumerated above, since its administrator can rewrite history undetectably. The roughly half-minute added per run by on-chain recording is therefore the price of removing the trusted intermediary altogether. Related blockchain-logging systems make the same exchange: the permissioned logging infrastructure of Putz et al. (2019) [18] and the multichain audit-trail service BlockTrail [19] trade some public verifiability for throughput, whereas the present design intentionally favours the strongest verifiability available.

Table 5 compares the proposed system with four related systems across eight capabilities. The general-purpose frameworks - AutoGen [3], CrewAI and LangGraph - provide neither a blockchain audit trail nor report generation, while the blockchain-governance architecture of Jan et al. (2025) [25] provides an audit trail but on a private, permissioned chain and through invasive integration. The proposed system is the only one to satisfy all eight criteria simultaneously.

DOI: https://doi.org/10.67317/IJMSRT26JUL026

Table 5: Feature Comparison with Related Systems

Feature	AutoGen [3]	CrewAI	LangGraph	Jan et al. [25]	Proposed
Stateful typed workflow	No	No	Yes	No	Yes
Research pipeline (9 agents)	No	Partial	No	No	Yes
HTML report generation	No	No	No	No	Yes
Blockchain audit trail	No	No	No	Yes	Yes
Public verifiability	No	No	No	No	Yes
Non-invasive hook	No	No	No	No	Yes
Open-source contract	No	No	No	No	Yes
100% task completion	N/A	N/A	N/A	N/A	Yes

Several limitations qualify these results. The evaluation uses five datasets of modest size, three of them synthetic, so the findings characterise small-to-moderate workloads rather than industrial-scale data, and the strong linear runtime trend may not extrapolate to datasets that are orders of magnitude larger. Quality scoring rests on two human raters and a four-dimension rubric and is therefore subject to the usual concerns of construct validity and limited sample size. The tamper experiments establish detection of payload modification but do not exercise adversarial behaviour against the network itself, which lies outside the threat model. The blockchain target is a local Ethereum-compatible test network; a production deployment would require migration to a public main network and an independent security audit of the smart contract of the kind motivated by the vulnerability literature [21, 22]. Finally, because LLM inference is stochastic, the absolute quality and runtime figures are expected to vary between runs; the trends, rather than the precise values, are the dependable findings.

4. Conclusion

An integrated architecture has been presented that couples a nine-agent autonomous research pipeline to a blockchain-based, immutable audit trail, addressing the absence of any tamper-evident, independently verifiable record of how an AI-generated result is produced. The key findings are summarised as follows:

- A nine-agent LangGraph pipeline with typed state, conditional routing and explicit termination guarantees completed 100% of tasks across five datasets spanning demographics, biology, finance, retail and sales, with a mean report-quality score of 4.12 out of 5.0 and strong inter-rater agreement (Cohen's kappa of 0.81).
- A non-invasive decorator hook (node_hook.py) conferred blockchain auditability on the entire workflow with two lines of integration code and no modification to any agent, and produced byte-identical reports with and without instrumentation.
- The append-only AgentAuditTrail smart contract recorded 73 agent actions with a mean confirmation time of 2.1 seconds and approximately 67,200 gas per action, adding only 6.2% to total runtime - well within the 10% design budget.
- All 50 controlled tamper experiments were detected through on-chain hash verification, a 100% detection rate.
- Verifiable provenance for an autonomous multi-agent pipeline is therefore attainable at a cost that does not compromise its usability, and the hook-plus-contract pattern generalises to any LangGraph-based workflow.

DOI: https://doi.org/10.67317/IJMSRT26JUL026

Future work will pursue Merkle-root batching of action digests to reduce the transaction count per run, storage of full payloads on content-addressed systems for fully decentralised provenance, zero-knowledge verification of agent outputs over confidential data, migration of the contract to a public production network following an independent security audit, and extension of the pipeline to unstructured inputs such as documents and images.

5. References

[1]Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (Vol. 30, pp. 5998-6010). Curran Associates.

[2]Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J. (2024). A survey on large language model based autonomous agents. In Frontiers of Computer Science (Vol. 18, No. 6, Article 186345). Springer.

[3]Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In arXiv preprint arXiv:2308.08155.

[4]Hong, S., Zheng, X., Chen, J., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., & Wu, C. (2024). MetaGPT: Meta programming for a multi-agent collaborative framework. In Proceedings of the International Conference on Learning Representations (ICLR).

[5]Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., & Zhang, X. (2024). Large language model based multi-agents: A survey of progress and challenges. In Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI) (pp. 8048-8057).

[6]Hutson, M. (2018). Artificial intelligence faces reproducibility crisis. In Science (Vol. 359, No. 6377, pp. 725-726). American Association for the Advancement of Science.

[7]Raji, I. D., Smart, A., White, R. N., Mitchell, M., Gebru, T., Hutchinson, B., Smith-Loud, J., Theron, D., & Barnes, P. (2020). Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT*) (pp. 33-44). ACM.

[8]Mokander, J., Schuett, J., Kirk, H. R., & Floridi, L. (2024). Auditing large language models: A three-layered approach. In AI and Ethics (Vol. 4, No. 4, pp. 1085-1115). Springer.

[9]European Parliament and Council of the European Union. (2024). Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). In Official Journal of the European Union (OJ L 2024/1689).

[10]Schneier, B., & Kelsey, J. (1999). Secure audit logs to support computer forensics. In ACM Transactions on Information and System Security (Vol. 2, No. 2, pp. 159-176). ACM.

[11]Crosby, S. A., & Wallach, D. S. (2009). Efficient data structures for tamper-evident logging. In Proceedings of the 18th USENIX Security Symposium (pp. 317-334). USENIX Association.

[12]Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: A survey. In International Journal of Web and Grid Services (Vol. 14, No. 4, pp. 352-375). Inderscience.

[13]Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart contracts for the Internet of Things. In IEEE Access (Vol. 4, pp. 2292-2303). IEEE.

[14]Wust, K., & Gervais, A. (2018). Do you need a blockchain? In Proceedings of the Crypto Valley Conference on Blockchain Technology (CVCBT) (pp. 45-54). IEEE.

[15]Liang, X., Shetty, S., Tosh, D., Kamhoua, C., Kwiat, K., & Njilla, L. (2017). ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) (pp. 468-477). IEEE.

[16]Ramachandran, A., & Kantarcioglu, M. (2018). SmartProvenance: A distributed, blockchain based data provenance system. In Proceedings of the 8th ACM Conference on Data and Application Security and Privacy (CODASPY) (pp. 35-42). ACM.

[17]Azaria, A., Ekblaw, A., Vieira, T., & Lippman, A. (2016). MedRec: Using blockchain for medical data access and permission management. In Proceedings of the 2nd International Conference on Open and Big Data (OBD) (pp. 25-30). IEEE.

DOI: https://doi.org/10.67317/IJMSRT26JUL026

[18]Putz, B., Menges, F., & Pernul, G. (2019). A secure and auditable logging infrastructure based on a permissioned blockchain. In Computers and Security (Vol. 87, Article 101602). Elsevier.
[19]Ahmad, A., Saad, M., Njilla, L., Kamhoua, C., Bassiouni, M., & Mohaisen, A. (2019). BlockTrail: A scalable multichain solution for blockchain-based audit trails. In Proceedings of the IEEE International Conference on Communications (ICC) (pp. 1-6). IEEE.
[20]Cohen, J. (1960). A coefficient of agreement for nominal scales. In Educational and Psychological Measurement (Vol. 20, No. 1, pp. 37-46). Sage.
[21]Luu, L., Chu, D. H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS) (pp. 254-269). ACM.
[22]Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A survey of attacks on Ethereum smart contracts (SoK). In Principles of Security and Trust (POST), Lecture Notes in Computer Science (Vol. 10204, pp. 164-186). Springer.
[23]Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2013). Keccak. In Advances in Cryptology - EUROCRYPT 2013, Lecture Notes in Computer Science (Vol. 7881, pp. 313-314). Springer.
[24]Merkle, R. C. (1988). A digital signature based on a conventional encryption function. In Advances in Cryptology - CRYPTO 87, Lecture Notes in Computer Science (Vol. 293, pp. 369-378). Springer.
[25]Jan, S., Razzaqi, H. A., Akarma, A., & Belgaum, M. R. (2025). A blockchain-monitored agentic AI architecture for trusted perception-reasoning-action pipelines. In Proceedings of the IEEE International Conference on Computing and Applications (ICCA). IEEE.

DOI: https://doi.org/10.67317/IJMSRT26JUL026