



(RESEARCH ARTICLE)



AGENTIC AI-DRIVEN DATA ENGINEERING: A FRAMEWORK FOR SELF-DIRECTED ENTERPRISE DATA PIPELINES

Anil Kumar Kolla ^{1,*} and Ameya Kokate ²

¹ Department of Information Technology, HonorHealth, Scottsdale, Arizona, USA.

² Department of Data Science and Analytics, HonorHealth, Scottsdale, Arizona, USA.

* Corresponding Author

ORCID Details

Anil Kumar Kolla: <https://orcid.org/0009-0001-8541-0855>

Ameya Kokate: <https://orcid.org/0009-0005-9777-3273>

International Journal of Frontline Research in Engineering and Technology, 2026, 06(01), 001–008

Publication history: Received on 14 August 2026; revised on 22 September 2026; accepted on 24 September 2026

Article DOI: <https://doi.org/10.56355/ijfret.2026.6.1.0015>

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

Enterprise data pipelines have grown in scale and interdependency faster than the operational practices used to maintain them. Pipeline failures are still predominantly diagnosed and remediated by human engineers reading logs, tracing lineage, and applying tacit knowledge, which produces long mean-time-to-recovery and concentrates institutional knowledge in a small number of individuals. This paper proposes an agentic architecture in which large language model (LLM) agents, constrained by explicit tool interfaces and policy guardrails, perform continuous monitoring, root-cause triage, and bounded remediation of production data pipelines. The framework specifies five cooperating agent roles — Observer, Diagnostician, Remediator, Verifier, and Orchestrator — operating over a shared state store containing pipeline metadata, lineage graphs, data-quality expectations, and incident history. We define a capability-tiering model that maps each remediation action to one of four autonomy levels (observe, recommend, act-with-approval, act-autonomously), and we describe a human-in-the-loop governance layer providing approval routing, immutable action logging, and rollback. We further propose an evaluation protocol covering task success rate, mean-time-to-detection, mean-time-to-recovery, false-remediation rate, and escalation precision, together with a reference workload derived from common ingestion, transformation, and schema-evolution failure modes. The paper concludes by analysing failure modes specific to agentic operation — hallucinated root causes, unsafe tool invocation, reward-free drift, and accountability gaps — and argues that autonomy must be earned incrementally through measured reliability rather than granted by architectural default.

Keywords: Agentic Systems, Large Language Models, Data Engineering, DataOps, Pipeline Orchestration, Human-in-the-Loop, Data Reliability, Lakehouse.

1 INTRODUCTION

The modern enterprise data platform is a dense graph of dependencies. A single analytical dashboard may depend on dozens of upstream transformations, each of which depends on ingestion jobs bound to external systems whose

schemas, availability, and semantics are outside the platform team's control. As the graph grows, the probability that some part of it is broken at any given moment approaches one.

Operational practice has not kept pace with this growth. When a pipeline fails, the prevailing response is still manual: an on-call engineer receives an alert, opens a log viewer, reconstructs the execution context, forms a hypothesis, and applies a fix. The steps are cognitively demanding but rarely novel. Empirically, a large fraction of production incidents resolve to a small set of recurring causes — upstream schema drift, late-arriving or partial data, credential expiry, resource exhaustion, and idempotency violations on retry. The work is repetitive without being mechanical, which is precisely the profile of tasks where language-model agents have shown promise [1], [2].

Recent work on tool-using and self-reflective LLM agents [1]–[4] demonstrates that models can decompose goals, invoke external functions, observe results, and revise plans. Applying this capability to data operations is attractive but non-trivial. Data pipelines are stateful, side-effecting, and frequently subject to regulatory constraint; an agent that "helpfully" backfills a partition or alters a schema can cause damage that is expensive or impossible to reverse. The engineering problem is therefore not whether an agent can act, but how to bound what it may act upon.

This paper makes four contributions:

- A reference architecture for agentic data engineering that separates perception, reasoning, action, and verification across distinct agent roles with distinct privileges.
- A capability-tiering model that binds each remediation primitive to an autonomy level and an approval policy, making the blast radius of agent action explicit and auditable.
- A governance layer specifying approval routing, immutable action provenance, and rollback semantics, designed to satisfy enterprise audit expectations and to align with recognised AI risk-management guidance [5].
- An evaluation protocol and reference failure workload enabling reproducible comparison between agentic, rule-based, and manual incident response.

1.1 Contribution type and scope

This is a design and evaluation-methodology contribution. It specifies an architecture and a measurement protocol; it does not report an empirical evaluation of a deployed system, and no performance claim should be read into it. Section 5 defines the experimental design, the baselines, and the reporting format in enough detail to be executed by an implementer, and Table 2 gives the pre-specified reporting template. The framework's claims are design arguments offered for testing, not validated results.

The remainder of the paper is organised as follows. Section 2 reviews related work. Section 3 presents the architecture. Section 4 describes implementation on a lakehouse platform. Section 5 specifies the evaluation protocol. Section 6 discusses limitations and failure modes. Section 7 concludes.

2 BACKGROUND AND RELATED WORK

2.1 Data Reliability Engineering

The recognition that data systems require reliability engineering comparable to that applied to services is relatively recent. Sculley *et al.* [6] characterised the hidden maintenance burden of machine learning systems, much of which originates in data dependencies rather than in models. Polyzotis *et al.* [7] catalogued data management challenges specific to production ML, and Breck *et al.* [8] proposed a validation regime for detecting data errors before they propagate into training and serving. These works establish the diagnostic vocabulary — schema skew, distribution skew, feature drift — that an automated agent must be able to reason about.

Operational practice in this area draws heavily on site reliability engineering [9], particularly the notions of error budgets, blameless postmortems, and progressive automation of toil. The framework presented here adopts the SRE stance that automation should absorb well-understood work while escalating genuine novelty.

2.2 Lakehouse Platforms and Metadata

The lakehouse model [10] unifies warehouse-style transactional guarantees with data-lake economics, with transactional table formats such as Delta Lake [11] providing ACID semantics, schema enforcement, and time travel over object storage. These properties matter for agentic operation for a specific reason: they make many remediations reversible. Time travel and atomic commits mean an agent's write can be rolled back deterministically, which is a precondition for granting it any write authority at all. Distributed execution engines such as Apache Spark [12] supply

the compute substrate and the execution metadata (task-level metrics, stage failures, skew indicators) on which diagnosis depends.

2.3 LLM Agents

The agentic paradigm couples a language model with external tools and an observe–reason–act loop. ReAct [1] interleaves reasoning traces with tool calls; Toolformer [3] shows models can learn tool invocation; Reflexion [2] adds verbal self-critique across attempts. Surveys [4], [13] describe the emerging design space of planning, memory, and multi-agent coordination. Retrieval augmentation [14] grounds generation in retrieved documents and is directly applicable here: runbooks, prior incident records, and lineage metadata form the retrieval corpus.

The principal known weakness is hallucination — fluent output unsupported by evidence [15]. In conversational settings this is an accuracy problem; in an agentic operations setting it is a safety problem, because a confidently wrong root cause can drive a destructive action. This observation motivates the separation of the Diagnostician and Verifier roles in Section 3.

2.4 Gap

Existing literature treats agents largely in software-engineering or general task-completion settings. Data platform operations differ in three respects: actions are frequently irreversible at the semantic level even when reversible at the storage level; correctness is often only observable downstream and with delay; and the regulatory surface (retention, residency, access) constrains which data an agent may even read. We are not aware of a published framework that binds LLM agency to a graduated autonomy model with explicit rollback and audit semantics for data pipelines. This paper addresses that gap.

3 PROPOSED FRAMEWORK

3.1 Design Principles

The architecture rests on five principles.

- **P1 — Least authority.** Each agent receives the narrowest tool set sufficient for its role. The Diagnostician has read-only access; only the Remediator holds write-capable tools, and only within its tier.
- **P2 — Evidence before action.** No remediation may be proposed without a citation set: the specific log lines, metrics, lineage edges, or prior incidents supporting the hypothesis. Uncited hypotheses are rejected by the Verifier.
- **P3 — Reversibility by default.** Actions are preferentially expressed as transactions on versioned tables so that rollback is a single operation.
- **P4 — Graduated autonomy.** Authority is a function of demonstrated reliability on a given action class, not a global setting.
- **P5 — Complete provenance.** Every observation, hypothesis, approval, action, and outcome is written to an append-only ledger sufficient to reconstruct the incident.

3.2 Agent Roles

- **Observer.** Subscribes to orchestrator events, job telemetry, and data-quality assertion results. It normalises heterogeneous signals into a canonical incident record and performs deduplication and correlation, suppressing the alert storms that arise when one upstream failure cascades. The Observer emits incidents; it does not interpret them.
- **Diagnostician.** Consumes an incident record and produces a ranked set of candidate root causes, each with an evidence citation set and a calibrated confidence. It has read access to logs, execution metrics, table history, the lineage graph, schema registry, and a retrieval index over prior incidents and runbooks. It is explicitly prohibited from proposing actions, which prevents the common failure in which a model rationalises a diagnosis backwards from a convenient fix.
- **Remediator.** Consumes a validated diagnosis and selects a remediation from a registry of typed, parameterised primitives — not free-form code. Representative primitives include retry with backoff, partition backfill over a bounded window, quarantine of failing records to a dead-letter table, schema evolution application, compute-resource adjustment, credential refresh trigger, and dependency-aware downstream pause. Each primitive declares its preconditions, its blast radius, its rollback procedure, and its autonomy tier.
- **Verifier.** Performs two checks. *Pre-action*, it validates that the diagnosis is evidentially supported, that the selected primitive's preconditions hold, and that the blast radius falls within policy. *Post-action*, it confirms that the target symptom cleared, that data-quality expectations on affected tables pass, and that no new anomaly

appeared in downstream dependents within a defined observation window. Failure of post-action verification triggers automatic rollback and escalation.

- **Orchestrator.** Sequences the above, manages timeouts and retries, routes approvals to the correct human owner based on the lineage-derived ownership map, and enforces circuit breakers — for example, halting all autonomous action for a domain after n verification failures within a window.

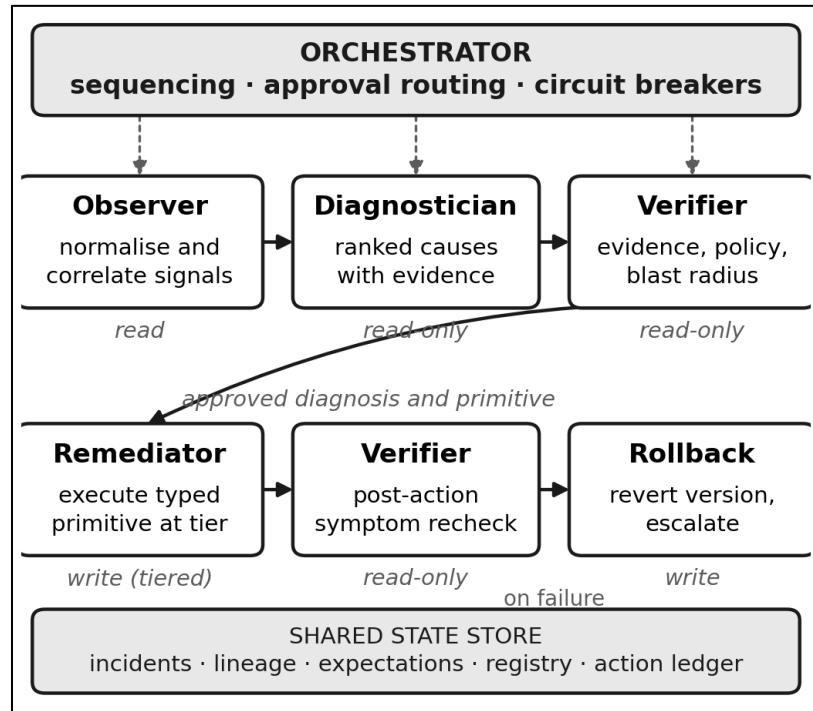


Figure 1: Reference architecture. Agent roles are separated by privilege; only the Remediator holds write-capable tools, and every action is verified before and after execution.

3.3 Capability Tiering

Each remediation primitive is assigned to one of four tiers, as summarised in Table 1:

Table 1: Capability tiering. Each remediation primitive is bound to one of four autonomy levels; autonomy is an earned, revocable property within each data domain.

Tier	Name	Agent authority	Example primitives
T0	Observe	Detect and report only	Anomaly annotation, incident correlation
T1	Recommend	Produce a proposed action for human execution	Schema change proposal, cost-optimisation suggestion
T2	Act with approval	Execute after explicit human approval	Bounded backfill, schema evolution, resource resizing
T3	Act autonomously	Execute without prior approval, with post-hoc notification	Transient-failure retry, record quarantine, credential refresh

Promotion between tiers is governed by measured performance rather than judgement. A primitive is eligible for promotion from T2 to T3 only after a defined number of consecutive successful executions with post-action verification passing and zero human overrides, evaluated per data domain rather than globally. Demotion is immediate on verification failure. This makes autonomy an earned, revocable property with an auditable basis.

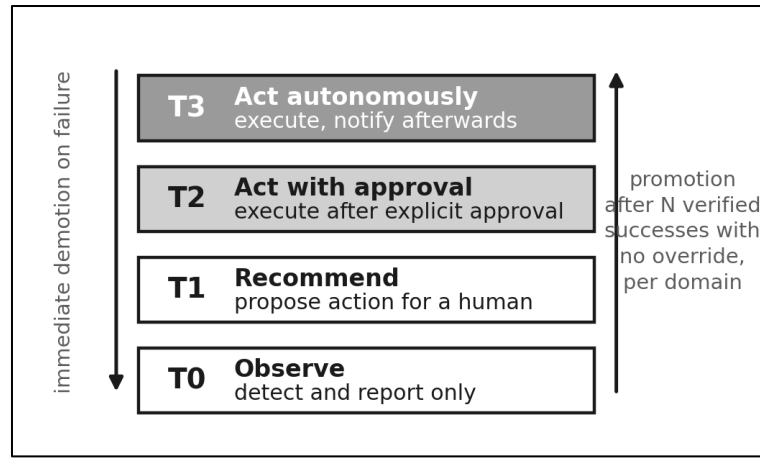


Figure 2: Capability tiering. Autonomy is an earned, revocable property of each remediation primitive within each data domain.

3.4 Shared State

Agents communicate through a shared state store rather than direct message passing, which keeps the incident record as the single source of truth and simplifies audit. The store holds: the incident record; the lineage graph including column-level edges where available; the expectation catalogue defining data-quality assertions per table; the primitive registry; the approval policy; and the append-only action ledger.

3.5 Human-in-the-Loop Governance

Approval routing uses the lineage graph to identify the owning steward of every affected asset, so approval requests reach the person with the relevant context rather than a generic queue. Approval requests present the diagnosis, the evidence citations, the proposed primitive with concrete parameters, the computed blast radius (tables, partitions, and downstream consumers affected), and the rollback plan. Time-boxed approvals expire into escalation rather than into silent inaction.

The design aligns with the govern-map-measure-manage structure of the NIST AI Risk Management Framework [5]: the primitive registry and tier assignment constitute *map*; the evaluation metrics of Section 5 constitute *measure*; the approval, circuit-breaker, and rollback mechanisms constitute *manage*.

4 IMPLEMENTATION CONSIDERATIONS

A reference implementation targets a cloud lakehouse. Orchestration events and job telemetry are consumed from the workflow scheduler's event API. Table history, schema, and time-travel operations are provided by the transactional table format [11]. Lineage is obtained from the platform catalogue where available and augmented by static analysis of transformation SQL for column-level edges. Data-quality assertions are expressed declaratively and evaluated as part of transformation execution, following the validation approach of Breck *et al.* [8].

Tools exposed to agents are implemented as typed functions with input validation, timeout enforcement, and mandatory dry-run modes for all write-capable operations. The dry-run requirement is significant: the Verifier evaluates the dry-run output — rows affected, partitions touched, schema delta — against policy before any real execution is permitted. Model outputs are constrained to a schema for structured decisions; free-form generation is used only for human-readable explanation, never for action selection.

Retrieval augmentation [14] indexes runbooks, architectural decision records, prior incident postmortems, and table documentation. Prior incidents are the highest-value corpus, since recurrence is common and a resolved incident is effectively a labelled example.

Cost and latency require attention. Invoking a large model on every telemetry event is prohibitive; the Observer therefore performs deterministic filtering and correlation, and model invocation begins only at the Diagnostician stage on deduplicated incidents.

5 EVALUATION PROTOCOL

We propose the following protocol so that results across implementations are comparable. *All quantitative values reported by an implementer must be measured on that implementer's own workload; this paper specifies the measurement, not the outcome.*

5.1 Reference Failure Workload

A fault-injection suite covering the recurring failure classes identified in Section 1: (i) additive and breaking upstream schema change; (ii) late-arriving and partial source data; (iii) duplicate delivery violating idempotency; (iv) credential and token expiry; (v) resource exhaustion and executor loss; (vi) referential integrity violation between dimension and fact loads; (vii) upstream silent semantic change (unit or encoding change with stable schema). Class (vii) is included deliberately as a negative control, since it is not detectable from execution telemetry alone and should produce escalation rather than remediation.

5.2 Metrics

- **Mean time to detection (MTTD)** — incident onset to Observer emission.
- **Mean time to recovery (MTTR)** — incident onset to verified resolution.
- **Diagnostic accuracy** — proportion of incidents whose top-ranked root cause matches ground truth.
- **Remediation success rate** — proportion of executed actions passing post-action verification.
- **False-remediation rate** — proportion of executed actions that resolve the symptom while leaving the underlying fault, or that introduce a new downstream anomaly. This is the most safety-relevant metric and should be reported separately from success rate.
- **Escalation precision and recall** — correctness of the decision to hand off to a human, measured against the negative-control class.
- **Human intervention rate** — approvals and overrides per incident, tracked over time as an indicator of trust calibration.
- **Cost per incident** — inference and compute cost, compared against the labour cost baseline.

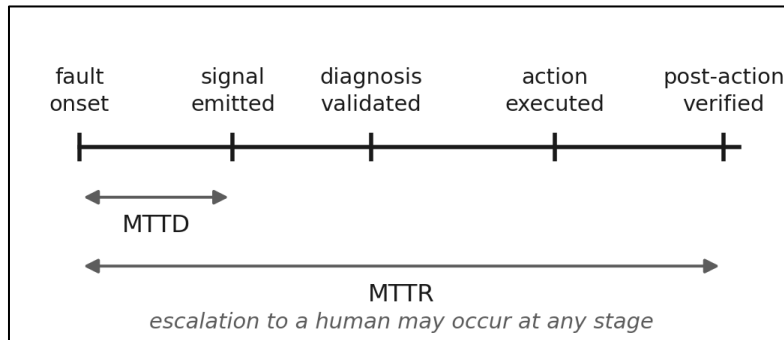


Figure 3: Incident timeline showing the intervals over which mean time to detection and mean time to recovery are measured.

5.3 Baselines

Three comparators: (i) manual on-call response; (ii) static rule-based automation without a language model; (iii) a single-agent LLM configuration without role separation or tiering. Comparator (iii) isolates the contribution of the architecture from the contribution of the model.

5.4 Pre-Specified Reporting Template

Results should be reported in the form of Table 2, per failure class and then in aggregate, so that comparisons across implementations are legible. The table is presented here unpopulated: the values are properties of a particular implementation and workload, and this paper does not assert them.

Table 2: Pre-specified reporting template. Remediation success, escalation precision and recall, human intervention rate, and cost per incident should be reported alongside in the same form. Confidence intervals over repeated fault-injection runs should accompany every mean.

Configuration	MTTD	MTTR	Diagnostic accuracy	False remediation
Manual on-call baseline	—	—	—	—
Rule-based automation	—	—	—	—
Single-agent LLM, no roles	—	—	—	—
Proposed architecture	—	—	—	—

5.5 Reporting

Implementers should report per-failure-class results rather than aggregates, since aggregate success rates conceal the distinction between trivially retryable transient faults and semantically complex ones. Confidence intervals over repeated injection runs should accompany all reported means.

6 DISCUSSION AND LIMITATIONS

- **Hallucinated diagnosis.** The dominant risk. A model may generate a plausible causal narrative unsupported by the telemetry [15]. The citation requirement (P2) and the independent Verifier mitigate but do not eliminate this; adversarial evaluation, in which incidents are injected with misleading correlated signals, should be part of any deployment assessment.
- **Reversibility is shallower than it appears.** Time travel restores table state, but it does not recall a dashboard a clinician or executive already acted upon, nor reverse a downstream export to an external system. Blast-radius computation must therefore include egress boundaries, and any primitive whose downstream consumers include external systems should be barred from T3 regardless of measured reliability.
- **Accountability.** When an autonomous action causes harm, responsibility must attach to an identifiable human owner. The framework assigns accountability to the steward who owns the affected asset and who controls its tier assignments; the action ledger exists to make this determinable after the fact. Organisations should treat tier promotion as a governance decision requiring the same rigour as a production change approval.
- **Distribution shift in the agent itself.** Model providers update models. A primitive promoted to T3 on the basis of measured reliability under one model version has no guaranteed reliability under another. Tier assignments should therefore be versioned against the model and re-validated on model change.
- **Scope.** The framework addresses operational faults. It does not address business-logic errors, in which a pipeline executes correctly but implements the wrong requirement; these remain firmly human territory and should not be presented as within agentic scope.
- **Evaluation validity.** Fault injection approximates but does not reproduce production failure distributions, which are long-tailed. Reported results should be understood as a lower bound on difficulty.

7 CONCLUSION AND FUTURE WORK

This paper proposed an architecture for applying LLM agents to enterprise data pipeline operations under explicit constraint. Its central claim is structural rather than empirical: that separating diagnosis from remediation, binding actions to a typed registry with declared blast radius and rollback, and tying autonomy to measured per-primitive reliability produces a system whose failure modes are bounded and auditable, in contrast to a monolithic agent granted broad tool access.

Future work should address three directions. First, calibration — models are poorly calibrated in confidence, and tier promotion depends on trustworthy confidence estimates. Second, multi-domain coordination, where incidents span ownership boundaries and approval routing becomes a negotiation rather than a lookup. Third, longitudinal study of trust calibration in operating teams, since the practical value of the system depends less on peak accuracy than on whether engineers correctly identify the cases in which it should not be trusted.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Disclosure of conflict of interest

The authors declare no conflicts of interest regarding this manuscript.

REFERENCES

- [1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.
- [2] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [3] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [4] L. Wang *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, 2024.
- [5] National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, Gaithersburg, MD, USA, Jan. 2023.
- [6] D. Sculley *et al.*, "Hidden technical debt in machine learning systems," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 2503–2511.
- [7] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2017, pp. 1723–1726.
- [8] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data validation for machine learning," in *Proc. Conf. Systems and Machine Learning (SysML/MLSys)*, 2019.
- [9] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, 2016.
- [10] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *Proc. Conf. Innovative Data Systems Research (CIDR)*, 2021.
- [11] M. Armbrust *et al.*, "Delta Lake: High-performance ACID table storage over cloud object stores," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.
- [12] M. Zaharia *et al.*, "Apache Spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.
- [13] Z. Xi *et al.*, "The rise and potential of large language model based agents: A survey," arXiv:2309.07864, 2023.
- [14] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [15] Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.