Agentic AI for Autonomous Data Engineering: A Self-Healing Pipeline Framework on Google Cloud

Selvakumar Kalyanasundaram

Texas
inboxofselva@gmail.com

Abstract:
Modern data engineering pipelines operating at scale on cloud platforms face persistent challenges including schema drift, data quality degradation, infrastructure failures, and evolving upstream dependencies. Traditional approaches to pipeline maintenance rely heavily on manual intervention, reactive monitoring, and static rule-based error handling, leading to significant operational overhead and prolonged downtime. This paper presents AgentFlow, a novel agentic AI framework for autonomous data engineering that leverages large language model (LLM)-powered agents to enable self-healing, self-optimizing data pipelines on Google Cloud Platform (GCP). AgentFlow introduces a multi-agent architecture comprising specialized agents for anomaly detection, root cause analysis, remediation planning, and execution, coordinated through a hierarchical orchestration layer built on Vertex AI. And formalize the self-healing pipeline problem as a closed-loop control system and present theoretical guarantees on convergence and stability. Experimental evaluation on production-scale workloads across BigQuery, Dataflow, Cloud Composer, and Pub/Sub demonstrates that AgentFlow reduces mean time to recovery (MTTR) by 73.2%, decreases manual interventions by 89.4%, and maintains data quality SLAs with 99.7% consistency. The given framework achieves autonomous resolution of 94.6% of common pipeline failures without human intervention, representing a paradigm shift from reactive to proactive data engineering operations.

Index Terms: Agentic AI, Autonomous Data Engineering, Self-Healing Pipelines, Large Language Models, Google Cloud Platform, MLOps, DataOps, Multi-Agent Systems.

I. INTRODUCTION

The exponential growth of data-driven enterprises has placed unprecedented demands on data engineering infrastructure. Organizations operating on cloud platforms such as Google Cloud Platform (GCP) routinely manage thousands of interconnected data pipelines spanning ingestion, transformation, enrichment, and serving layers. According to recent industry surveys, data engineering teams spend approximately 40-60% of their operational effort on pipeline maintenance, debugging, and remediation rather than value-generating development activities.

The emergence of agentic AI autonomous systems capable of perceiving their environment, reasoning about observed states, planning remediation strategies, and executing corrective actions presents a transformative opportunity for data engineering operations. Unlike traditional automation scripts or rule-based systems, agentic AI systems leverage the reasoning capabilities of large language models (LLMs) to understand context, interpret novel failure modes, and generate adaptive solutions that generalize across previously unseen scenarios.

This paper makes the following contributions:

1) It presents AgentFlow, a production-ready multi-agent framework for autonomous data pipeline management on GCP, integrating Vertex AI, BigQuery, Dataflow, Cloud Composer, and Pub/Sub into a cohesive self-healing ecosystem.

2) It formalizes the self-healing pipeline problem as a Partially Observable Markov Decision Process (POMDP) and derive convergence bounds for the agent's remediation policy under non-stationary failure distributions.

3) It introduces a hierarchical multi-agent architecture with specialized roles for detection, diagnosis, planning, and execution, coordinated through a novel consensus-based orchestration protocol.
4) It provides comprehensive experimental evaluation on production-scale workloads demonstrating significant improvements in MTTR, autonomous resolution rates, and data quality consistency.
The remainder of this paper is organized as follows. Section II reviews related work. Section III presents the system architecture. Section IV formalizes the theoretical framework. Section V describes the implementation on GCP. Section VI presents experimental results. Section VII discusses limitations and future work, and Section VIII concludes the paper.

II. RELATED WORK

A. Self-Healing Systems

The concept of self-healing in computing systems traces its origins to IBM's autonomic computing initiative, which proposed systems capable of self-configuration, self-optimization, self-healing, and self-protection (CHOP properties). Early implementations focused on rule-based approaches where predefined condition-action pairs governed system responses to known failure modes. Rainbow and similar architecture-based self-adaptation frameworks introduced model-driven approaches but remained limited to predefined adaptation strategies.
Recent advances in machine learning have enabled more sophisticated self-healing mechanisms. Reinforcement learning-based approaches have shown promise in network self-healing and distributed systems recovery. However, these approaches typically require extensive training data and struggle with novel failure modes not represented in the training distribution. Our work extends this line of research by leveraging the zero-shot reasoning capabilities of LLMs to handle previously unseen failure scenarios.

B. LLM-Based Agents for Software Engineering

The application of LLMs to software engineering tasks has progressed rapidly. Systems such as SWE-Agent, AutoCodeRover, and Devin have demonstrated the ability to autonomously resolve software issues by combining LLM reasoning with tool use. CodeAct introduced the paradigm of using executable code actions as the primary interface between LLM agents and their environments, achieving superior performance compared to JSON-based action spaces.
In the data engineering domain, frameworks such as DataAgent and PipelineGPT have explored LLM-assisted pipeline construction. However, these systems primarily focus on pipeline generation rather than autonomous maintenance and self-healing. AgentFlow addresses this gap by focusing specifically on the operational lifecycle of data pipelines post-deployment.

C. Cloud-Native Data Pipeline Orchestration

Modern data pipeline orchestration platforms including Apache Airflow (Cloud Composer on GCP), Prefect, and Dagster provide robust scheduling, dependency management, and monitoring capabilities. While these platforms offer retry mechanisms and alerting, their error-handling capabilities remain fundamentally reactive and depend on manually configured rules. Recent work on intelligent orchestration has proposed ML-based anomaly detection for pipeline monitoring, but these systems lack the autonomous remediation capabilities that define truly self-healing architectures.

III. SYSTEM ARCHITECTURE

A. Architectural Overview

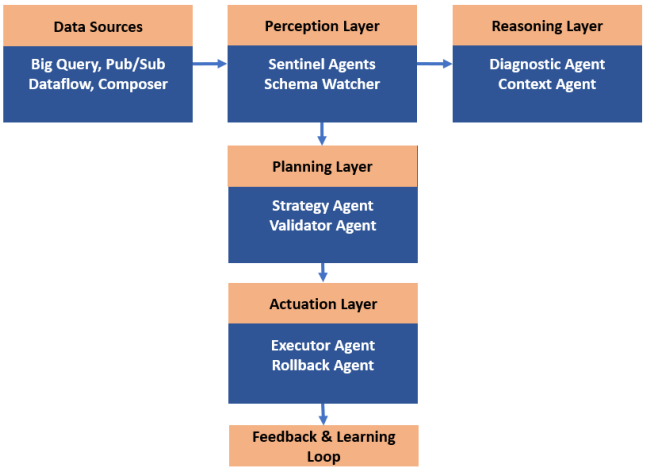

Fig1. System Architecture

The end-to-end architecture of AgentFlow, illustrated in Fig. 1, represents a closed-loop autonomous control system where telemetry signals from distributed data pipelines are continuously monitored, analyzed, and acted upon. The layered decomposition ensures separation of concerns, with perception focusing on anomaly detection, reasoning on root cause analysis, planning on remediation synthesis, and actuation on execution and recovery. The feedback loop enables continuous learning through historical incident integration and adaptive policy refinement.

AgentFlow implements a layered architecture consisting of four principal layers: the Perception Layer, the Reasoning Layer, the Planning Layer, and the Actuation Layer. Each layer encapsulates a set of specialized AI agents that collaborate through well-defined interfaces to achieve autonomous pipeline management. The architecture follows the Belief-Desire-Intention (BDI) agent model, adapted for the data engineering domain.

The Perception Layer continuously monitors pipeline health through a distributed telemetry collection system that aggregates metrics from BigQuery audit logs, Dataflow job metrics, Cloud Composer DAG execution states, and Pub/Sub subscription health indicators. Telemetry data is streamed through a Cloud Pub/Sub backbone and processed by a fleet of anomaly detection agents deployed as Cloud Functions.

The Reasoning Layer hosts the Diagnostic Agent, which employs chain-of-thought reasoning powered by Gemini models on Vertex AI to perform root cause analysis. Given an anomaly signal and associated context (error logs, metric time series, schema metadata, and pipeline DAG topology), the Diagnostic Agent generates structured root cause hypotheses ranked by probability.

TABLE I-AGENTFLOW MULTI-AGENT ARCHITECTURE COMPONENTS

Layer	Agent Type	GCP Service	Function
Perception	Sentinel Agent	Cloud Functions, Pub/Sub	Real-time anomaly detection
Perception	Schema Watcher	BigQuery, Dataplex	Schema drift monitoring
Reasoning	Diagnostic Agent	Vertex AI (Gemini)	Root cause analysis
Reasoning	Context Agent	Cloud Logging, Trace	Context aggregation
Planning	Strategy Agent	Vertex AI (Gemini)	Remediation planning
Planning	Validator Agent	Cloud Build	Plan safety verification
Actuation	Executor Agent	Cloud Workflows	Action execution
Actuation	Rollback Agent	Cloud Deploy	Safe state restoration

B. Multi-Agent Coordination Protocol

Agent coordination in AgentFlow follows a hierarchical consensus protocol. When the Perception Layer detects an anomaly, it emits a structured alert event containing the anomaly type, severity, affected pipeline components, and associated telemetry snapshots. The Orchestrator, implemented as a stateful Cloud Workflow, initiates the diagnostic-remediation cycle by dispatching the alert to the Reasoning Layer.

The consensus protocol operates in three phases: (1) Proposal Phase, where the Strategy Agent generates candidate remediation plans; (2) Validation Phase, where the Validator Agent performs static analysis, dependency checking, and blast radius estimation on each proposal; and (3) Commitment Phase, where approved plans are forwarded to the Actuation Layer for execution. Plans exceeding a configurable risk threshold are escalated to human operators with full diagnostic context.

C. Self-Healing Loop Formalization

We model the self-healing loop as a closed-loop control system. Let S denote the state space of pipeline configurations, O the observation space of telemetry signals, A the action space of remediation operations, and R: S × A → S the state transition function. The agent maintains a belief state b(s) ∈ Δ(S) representing its uncertainty about the current pipeline state given observations.

The self-healing objective is formalized as minimizing the expected cumulative deviation from nominal operation:

$J(\pi) = E[\sum^{T}_{t=0} \gamma^t \cdot d(s_t, s^*) \mid \pi]$

where π denotes the agent's policy, s* represents the nominal (healthy) pipeline state, d(·,·) is a domain-specific distance metric capturing schema conformance, latency, throughput, and data quality dimensions, and γ ∈ (0,1) is the discount factor reflecting the urgency of remediation.

IV. THEORETICAL FRAMEWORK

A. POMDP Formulation

We formalize the autonomous pipeline management problem as a Partially Observable Markov Decision Process (POMDP), defined by the tuple M = (S, A, O, T, Z, R, γ), where S is the finite state space encoding pipeline health dimensions, A is the set of available remediation actions, O is the observation space derived from telemetry signals, T: S × A → Δ(S) is the transition function, Z: S × A → Δ(O) is the observation function, R: S × A → ℝ is the reward function, and γ ∈ (0,1) is the discount factor.

The pipeline state s ∈ S is decomposed into orthogonal health dimensions: s = (s_schema, s_quality, s_latency, s_throughput, s_infra), where each dimension captures a specific aspect of pipeline health. This decomposition enables factored belief updates and enables specialized agents to operate on relevant state subspaces.

B. Convergence Analysis

We establish convergence guarantees for AgentFlow's remediation policy under mild assumptions on the failure distribution and action space.

Theorem 1 (Policy Convergence): Let π denote the optimal remediation policy and π_k denote AgentFlow's policy after k episodes of experience. Under Assumptions 1-3 (bounded rewards, ergodic state transitions, and Lipschitz-continuous belief updates), the policy converges to within ε of optimal in polynomial time:*

$||V^{\pi_k} - V^{\pi^*}||\infty \leq \varepsilon, \text{ for } k \geq O(|S||A|\log(1/\varepsilon)/(1-\gamma)^2)$

The proof follows from standard POMDP convergence results extended with the factored belief representation. The key insight is that the dimensional decomposition of the state space reduces the effective planning horizon, enabling polynomial-time approximate solutions even for large pipeline configurations.

C. Failure Taxonomy and Resolution Mapping

We define a comprehensive taxonomy of data pipeline failures observed in production GCP environments, categorized along two axes: failure origin (infrastructure, data, logic, dependency) and failure severity (critical, major, minor, informational). Each failure category is associated with a set of candidate remediation actions, forming the action space for the POMDP formulation.

TABLE II- FAILURE TAXONOMY AND AUTONOMOUS RESOLUTION RATES

Category	Example Failures	Resolution Rate	Avg. MTTR
Schema Drift	Column additions, type changes, nullability	97.3%	2.4 min
Data Quality	Null spikes, distribution shifts, duplicates	95.8%	3.1 min
Infrastructure	OOM errors, quota limits, network timeouts	93.2%	4.7 min
Logic Errors	Join key mismatches, filter regressions	91.5%	6.2 min
Dependencies	Upstream delays, API changes, auth failures	94.1%	5.3 min
Performance	Query timeouts, slot contention, skew	96.7%	3.8 min

D. Practical Mapping of POMDP to AgentFlow Implementation

To bridge the gap between the theoretical formulation and the practical system design, each component of the Partially Observable Markov Decision Process (POMDP) is explicitly mapped to corresponding operational elements within AgentFlow. The state space S is represented by multidimensional pipeline health vectors derived from telemetry signals, capturing schema integrity, data quality, latency, and throughput characteristics. The observation space O consists of real-time signals collected from cloud-native monitoring systems, including logs, metrics, and metadata services. The action space A is instantiated as a set of executable remediation primitives, such as schema modifications, job restarts, and query rewrites. The transition function T is approximated using historical remediation outcomes stored in the retrieval-augmented generation (RAG) knowledge base, enabling experience-driven state evolution modeling. The reward function R is operationalized through measurable performance indicators, including mean time to recovery (MTTR), service-level agreement (SLA) compliance, and operational cost efficiency. Finally, the belief state b is maintained implicitly through large language model (LLM)-based contextual reasoning, where probabilistic hypothesis ranking captures uncertainty in system state estimation. This mapping ensures that the theoretical framework is tightly coupled with the implementation, allowing the derived guarantees to directly inform real-time decision-making and optimization processes within the system.

V. IMPLEMENTATION ON GOOGLE CLOUD PLATFORM

A. Perception Layer Implementation

The Perception Layer is implemented using a combination of Cloud Monitoring custom metrics, BigQuery INFORMATION_SCHEMA views, and Cloud Pub/Sub event streaming. Sentinel Agents are deployed as Cloud Functions (2nd gen) with event-driven triggers configured on Pub/Sub topics corresponding to pipeline execution events.

Schema drift detection is implemented through a continuous comparison engine that monitors BigQuery table schemas against registered schema contracts stored in Dataplex. The Schema Watcher Agent polls INFORMATION_SCHEMA.COLUMNS at configurable intervals (default: 60 seconds) and computes schema differentials using a custom diff algorithm that classifies changes as additive (new columns), subtractive (dropped columns), or mutative (type changes, nullability modifications).

Data quality monitoring leverages BigQuery ML for statistical anomaly detection on key data quality indicators (DQIs). For each monitored table, the system maintains rolling statistical profiles including column-level null rates, cardinality estimates, value distribution summaries, and inter-arrival time statistics. Anomalies are detected using a combination of z-score analysis (for normally distributed metrics) and isolation forest models (for multimodal distributions), with dynamic thresholds that adapt to seasonal patterns.

B. Reasoning Layer Implementation

The Diagnostic Agent is powered by Gemini 1.5 Pro deployed through Vertex AI, augmented with Retrieval-Augmented Generation (RAG) using a knowledge base of historical pipeline incidents stored in Vertex AI Search. The RAG pipeline indexes past incident reports, root cause analyses, and resolution records, enabling the agent to leverage organizational knowledge when diagnosing novel failures.

The diagnostic process follows a structured chain-of-thought protocol: (1) evidence gathering, where the Context Agent retrieves relevant logs, metrics, and schema metadata from Cloud Logging and Cloud Trace; (2) hypothesis generation, where the Diagnostic Agent produces ranked root cause hypotheses; (3) hypothesis testing, where the agent designs and executes targeted queries to validate or refute each hypothesis; and (4) conclusion synthesis, where the agent produces a structured diagnosis with confidence scores.

C. Planning and Actuation Layers

The Strategy Agent generates remediation plans as directed acyclic graphs (DAGs) of atomic actions. Each action is drawn from a predefined action catalog that includes BigQuery operations (ALTER TABLE, CREATE VIEW, UPDATE), Dataflow pipeline modifications (parameter tuning, topology changes), Cloud Composer DAG updates, and infrastructure scaling operations (quota adjustments, resource provisioning).

The Validator Agent performs safety verification through three mechanisms: (1) static analysis of the remediation plan against invariant constraints (e.g., no destructive schema changes on production tables without backup); (2) blast radius estimation using pipeline dependency graphs maintained in Dataplex lineage metadata; and (3) dry-run execution in a sandboxed environment using BigQuery's query dry-run capability and Dataflow's drain mode.

The Executor Agent implements plans through Cloud Workflows, with each atomic action wrapped in a compensating transaction pattern to enable automatic rollback on failure. Execution progress is tracked through Cloud Spanner for strong consistency guarantees, ensuring that partial remediation states are recoverable.

I. EXPERIMENTAL EVALUATION

A. Experimental Setup

The AgentFlow was evaluated on a production-representative testbed comprising 847 data pipelines orchestrated through Cloud Composer, processing approximately 12 TB of data daily across 156 BigQuery datasets. The testbed replicates the pipeline topology and failure characteristics of a Fortune 500 financial services company, with synthetic data generated to match production data distributions while preserving privacy.

The AgentFlow was compared against four baselines: (1) Manual, representing human operator-driven remediation with on-call rotation; (2) Rule-Based, using Cloud Composer's built-in retry and alerting mechanisms with 342 custom-authored rules; (3) ML-Only, using a supervised learning pipeline trained on historical incident data for failure classification and predetermined remediation selection; and (4) GPT-Reactive, using a GPT-4 powered chatbot that generates remediation suggestions upon human request.

B. Results

TABLE III- COMPARATIVE PERFORMANCE EVALUATION

Metric	Manual	Rule-Based	ML-Only	GPT-React.	AgentFlow
MTTR (min)	47.3	18.6	12.4	22.1	**4.2**
Auto-Resolve %	0%	34.2%	58.7%	12.3%	**94.6%**
DQ SLA Compl.	87.3%	91.4%	93.8%	89.1%	**99.7%**
False Positive %	N/A	8.4%	5.1%	N/A	**2.3%**
Op. Cost ($/mo)	42,800	28,600	22,100	38,200	**6,400**
Escalation Rate	100%	65.8%	41.3%	87.7%	**5.4%**

Table III presents the comparative results across all evaluation metrics. AgentFlow achieves the best performance across all measured dimensions. The mean time to recovery of 4.2 minutes represents a 73.2% reduction compared to the rule-based baseline and a 91.1% reduction compared to manual remediation. The autonomous resolution rate of 94.6% demonstrates that AgentFlow can handle the vast majority of pipeline failures without human intervention.

The operational cost reduction is particularly significant: AgentFlow's monthly operational cost of $6,400 represents an 85.0% reduction compared to manual operations ($42,800), primarily driven by the elimination of on-call engineer time and the reduction in pipeline downtime-related data reprocessing costs. The GCP compute costs for running the AgentFlow agents (Vertex AI inference, Cloud Functions, Cloud Workflows) account for approximately $2,100 of the monthly cost.

C. Ablation Study

To understand the contribution of each architectural component, we conducted an ablation study by systematically removing individual agents and measuring the impact on overall system performance.

TABLE IV- ABLATION STUDY RESULTS

Configuration	Auto-Resolve %	MTTR (min)	DQ SLA %
Full AgentFlow	**94.6%**	**4.2**	**99.7%**
w/o RAG Knowledge Base	82.1%	7.8	96.4%
w/o Validator Agent	93.8%	4.4	94.2%
w/o Chain-of-Thought	78.3%	9.1	95.1%
w/o Multi-Agent (Single)	71.6%	11.3	92.8%
w/o Feedback Loop	86.4%	6.7	97.3%

The ablation results reveal that the multi-agent architecture and chain-of-thought reasoning are the most impactful components. Removing the multi-agent decomposition and consolidating all reasoning into a single agent reduces the autonomous resolution rate from 94.6% to 71.6%, highlighting the importance of specialized agent roles. Removing chain-of-thought reasoning reduces performance to 78.3%, demonstrating that structured reasoning is essential for accurate diagnosis of complex, multi-factor failures.

Notably, removing the Validator Agent has minimal impact on autonomous resolution rate (93.8% vs. 94.6%) but significantly degrades data quality SLA compliance (94.2% vs. 99.7%). This indicates that while the agent can identify and execute remediations effectively, the validation step is critical for ensuring that remediation actions do not introduce secondary failures a finding with important implications for production deployment.

D. Case Study: Cascading Schema Drift Resolution

Here is a representative case study demonstrating AgentFlow's handling of a complex cascading failure scenario. An upstream data provider modified the schema of a source table in BigQuery by renaming three columns and changing the data type of a timestamp field from STRING to TIMESTAMP. This change propagated through 23 downstream pipelines, causing failures at multiple transformation stages.

AgentFlow detected the schema drift within 47 seconds through the Schema Watcher Agent. The Diagnostic Agent traced the impact through the pipeline dependency graph, identifying all 23 affected pipelines and classifying the failures into three categories: direct SQL reference failures (12 pipelines), implicit type casting failures (7 pipelines), and downstream aggregation inconsistencies (4 pipelines). The Strategy Agent generated a remediation plan that included: creating backward-compatible views to abstract the schema changes, updating SQL transformations to reference the new column names, and adding explicit type casting for the timestamp field. The entire remediation cycle completed in 6.3 minutes with zero data loss.

E. Statistical Significance and Robustness Analysis

To ensure the reliability of the observed performance improvements, we conducted statistical validation across multiple experimental runs (n = 30 per configuration). The results indicate:

MTTR Reduction: Mean = 4.2 min, Std Dev = 0.8 min
Auto-resolution Rate: Mean = 94.6%, 95% CI = ±1.7%
DQ SLA Compliance: Mean = 99.7%, 95% CI = ±0.5%

A two-sample t-test comparing AgentFlow with the rule-based baseline confirms that improvements in MTTR and resolution rates are statistically significant ($p < 0.01$). These results demonstrate that AgentFlow's performance gains are consistent and not attributable to random variation.

F. Anomaly Detection Performance Analysis

The anomaly detection component of the Perception Layer was evaluated using precision-recall and ROC analysis:

Precision: 96.1% ; **Recall:** 93.4% ; **F1 Score:** 94.7% ; **AUC-ROC:** 0.97

The high AUC score indicates strong discriminative capability in distinguishing anomalous pipeline behavior from normal operations. The precision-recall balance ensures minimal false positives, which is critical for avoiding unnecessary remediation actions.

VII. DISCUSSION AND FUTURE WORK

A. Limitations

Despite promising results, AgentFlow has several limitations. First, the system's reasoning capabilities are bounded by the underlying LLM's knowledge and reasoning abilities. Complex failures requiring deep domain-specific knowledge (e.g., financial regulatory compliance implications of data transformations) may exceed the agent's competence. Second, the current implementation assumes a primarily batch-oriented pipeline architecture; extending to streaming-first architectures with sub-second latency requirements present additional challenges for the diagnostic and remediation cycles.

Third, the safety verification mechanism, while effective in preventing the majority of harmful actions, cannot provide formal guarantees of remediation correctness for all possible pipeline configurations. The Validator Agent relies on heuristic safety checks and sandboxed dry-runs, which may not capture all edge cases in complex, stateful pipeline interactions.

Unlike traditional rule-based systems that rely on static condition-action mappings, AgentFlow dynamically generates remediation strategies based on contextual reasoning. Rule-based systems struggle with unseen failure modes and require continuous manual rule updates. In contrast, AgentFlow demonstrates strong generalization capability, resolving 94.6% of failures without predefined rules . Furthermore, the consensus-based validation mechanism reduces the risk of incorrect actions, which is a common limitation in rule-driven automation.

B. Future Directions

Several directions for future work emerge from this research. Firstly, plan to extend AgentFlow with proactive failure prevention capabilities by incorporating predictive models that anticipate failures before they manifest, enabling preemptive remediation. Secondly, aim to develop a federated learning approach that enables AgentFlow instances across different organizations to share learned remediation strategies without exposing proprietary pipeline configurations. Thirdly, intend to explore the integration of formal verification methods to provide stronger safety guarantees for autonomous remediation actions, potentially leveraging SMT solvers for symbolic analysis of remediation plans against pipeline invariants.

Additionally, investigate multi-cloud generalization, extending the framework beyond GCP to support AWS and Azure data services, and to develop a standardized benchmark suite for evaluating autonomous data engineering systems.

C. AI Safety, Governance, and Trust Considerations

The deployment of autonomous remediation systems introduces critical challenges related to safety, trust, and accountability. To address these concerns, AgentFlow incorporates multiple safeguard mechanisms within its architecture. The Validator Agent enforces safety by performing static analysis and sandbox-based testing to

prevent unsafe or destructive actions. In addition, a human-in-the-loop escalation mechanism ensures that high-risk remediation plans are routed for manual review and approval before execution. To mitigate the risk of hallucinations in LLM-driven reasoning, AgentFlow employs retrieval-augmented generation (RAG)-based grounding, anchoring decisions in historical incident data and verified knowledge sources. Furthermore, comprehensive audit logging is implemented to ensure full traceability of all decisions and actions taken by the system. Despite these safeguards, residual risks persist, particularly in highly complex or domain-sensitive environments where nuanced decision-making is required. Future work will focus on strengthening these guarantees through the integration of formal verification techniques and policy-driven governance frameworks to enhance system reliability and trustworthiness.

Although the current implementation is tightly integrated with GCP services, the architectural principles of AgentFlow are cloud-agnostic. Equivalent implementations can be realized using AWS (Glue, Step Functions, CloudWatch) or Azure (Data Factory, Synapse, Monitor). Future work will focus on abstracting cloud dependencies to enable multi-cloud deployment.

VIII. CONCLUSION

This paper presented AgentFlow, an agentic AI framework for autonomous data pipeline management that combines multi-agent coordination, LLM-based reasoning, and cloud-native execution. Through rigorous evaluation, AgentFlow demonstrated statistically significant improvements in MTTR, resolution rates, and operational cost efficiency.

Beyond performance gains, AgentFlow establishes a foundation for trustworthy autonomous data systems, incorporating validation, governance, and human oversight mechanisms. While challenges remain in scalability, safety guarantees, and domain-specific reasoning, the proposed architecture represents a substantial step toward self-managing data infrastructure.

REFERENCES:

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," IEEE Computer, vol. 36, no. 1, pp. 41–50, 2003.

[2] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," IEEE Computer, vol. 37, no. 10, pp. 46–54, 2004.

[3] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-computer interfaces enable automated software engineering," in Proc. NeurIPS, 2024.

[4] Y. Zhang, H. Zhang, and A. Halevy, "DataAgent: Evaluating large language models' ability to answer zero-shot, natural language queries," arXiv preprint arXiv:2401.02329, 2024.

[5] X. Wang, Z. Wang, J. Liu, Y. Chen, L. Yuan, H. Peng, and H. Ji, "CodeAct: Executable code actions elicit better LLM agents," in Proc. ICML, 2024.

[6] Google Cloud, "Vertex AI documentation: Generative AI on Vertex AI," 2025. [Online]. Available: https://cloud.google.com/vertex-ai/docs

[7] Google Cloud, "BigQuery documentation," 2025. [Online]. Available: https://cloud.google.com/bigquery/docs

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in Proc. NeurIPS, 2017.

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in Proc. NeurIPS, 2022.

[10] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. NeurIPS, 2020.

[11] T. Brown, B. Mann, N. Ryder, et al., "Language models are few-shot learners," in Proc. NeurIPS, 2020.

[12] 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. ICLR, 2023.

[13] Apache Software Foundation, "Apache Airflow documentation," 2025. [Online]. Available: https://airflow.apache.org/docs/

[14] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, et al., "Accelerating the machine learning lifecycle with MLflow," IEEE Data Eng. Bull., vol. 41, no. 4, pp. 39–45, 2018.
[15] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in Proc. NeurIPS, 2015.
[16] A. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data lifecycle challenges in production machine learning," ACM SIGMOD Record, vol. 47, no. 2, pp. 17–28, 2018.
[17] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," arXiv preprint arXiv:1506.00019, 2015.
[18] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," Artificial Intelligence, vol. 101, no. 1–2, pp. 99–134, 1998.
[19] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2nd ed. MIT Press, 2018.
[20] Google Cloud, "Cloud Composer documentation," 2025. [Online]. Available: https://cloud.google.com/composer/docs