International Journal of Latest Research in Engineering and Technology (IJLRET)
ISSN: 2454-5031
www.ijlret.com || Volume 11 - Issue 08 || August 2025 || PP. 78-83

Integrating AI-Powered Data-Lineage Agents with Graph Databases for Enhanced Pipeline Transparency

Abhishek Anand
Data Engineer II - Analytics, Grubhub Holdings Inc.
New York City, USA

Abstract: This paper answers with intelligent data-lineage agents based on artificial intelligence methods and graph databases, offering a way to transparency and accelerating ETL pipeline analysis. The volumes and heterogeneity of orchestration tools increase they are what make for so many invisible points of failure late duplicating computations lost time and money this study is important because it tries to show how best to bring together the latest AI agents with graph databases towards more complete timely lineage tracing thus reducing incident localization time while building a dependable foundation for analytics as well as machine learning. This approach enables every metric to be instantly traced to its source data and code versions. The innovation of the method described in this paper lies in hybrid algorithms: few-shot prompting with large language models to extract the semantics of SQL and Spark code, utilizing graph neural networks to infer hidden relationships in a property graph, and reinforcement-learning methods based on expert feedback for enhancing data quality. The agent architecture comprises an event-subscription module conforming to the OpenLineage specification, a component for linguistic analysis of code, a machine inference layer that validates relationships, and a Neo4j graph database driver that supports versioning. The main findings are that the combination of AI agents and graph databases allows multi-hop lineage queries to run in fractions of a second, rather than requiring many join operations; automatically captures and updates dependency versions, with no costly migrations; preserves and protects the business semantics of each node; and a user-feedback loop that eventually minimizes the need for manual corrections. This article will be of value to data specialists, analytical platform architects, and IT project managers interested in enhancing the transparency, reliability, and regulatory compliance of modern ETL pipelines.

Keywords: AI-based lineage agents, graph database, lineage, ETL pipelines, OpenLineage, property graph, graph neural networks, reinforcement learning, data transparency, systems integration

Introduction

The lack of transparency in these pipelines gives rise to invisible failure points: duplicated computations, delays in fault identification, and consequently diminished trust in analytical products. Data engineers are often compelled to react to consequences rather than create value, as empirical evidence suggests: according to IBM estimates, up to 80% of specialists' work time is spent on data discovery, cleansing, and preparation, whereas only one-fifth remains for actual analysis [1]. Direct financial losses are comparable to major IT budgets: Gartner estimates that poor data quality costs an average enterprise at least USD 12.9 million per year, excluding regulatory fines and indirect costs associated with missed opportunities and reputational risks [2]. The present article aims to demonstrate how the integration of intelligent lineage agents, based on state-of-the-art AI methods, with graph databases provides comprehensive and up-to-date lineage, shortens incident-resolution time, and establishes a robust foundation for analytics and machine learning, where each metric can be instantly linked to its source data and code versions.

Materials and Methodology

The study is based on the analysis of 15 sources, including academic articles, industry reports, case studies, and technical documentation. The theoretical foundation comprised works on lineage extraction using large-scale language models and hybrid algorithms: Li et al. demonstrated the effectiveness of few-shot prompting for parsing SQL and Spark code [9], Shi reviewed Text-to-SQL approaches for integrating business semantics [10], Holagh and Kobti described graph neural-network methods for predicting hidden relationships in a property graph [11], and Parra et al. presented reinforcement-learning techniques with expert feedback for data-quality improvement [12]. Additionally, survey and time-cost reports on data-preparation activities were considered: Mathur estimated that up to 80% of work time is devoted to ETL tasks [1], Gartner determined annual losses from poor data quality at USD 12.9 million for the average enterprise [2], Dataversity identified data-silo issues in 68% of companies [3], and TDWI reported that engineers spend two days per week on incident localization and resolution [5].

Methodologically, the study employed a combination of approaches, including comparative analysis of lineage-storage architectures, a systematic review of integration tools, and content analysis of survey data. In the comparative analysis, relational solutions were contrasted with property-graph models, drawing on the characteristics of graph databases Memgraph [7] and Neo4j, as well as the UBS case study on real-time lineage-query acceleration [15] and the Narayanasamy and Chen report on graph-store applications in analytics [6]. The systematic review covered the OpenLineage specification and its support in Apache Airflow [13], as well as guaranteed-delivery mechanisms for events via Kafka queues [14]. Content analysis of empirical data was based on the Emarketer survey on AI-agent integration and metadata-unification challenges [8].

Results and Discussion

Most existing lineage systems rely on individual-tool logs, resulting in provenance information being scattered across catalogs, ticket-tracking systems, and orchestration scripts. This forces engineers to manually correlate different records. The more components there are, the weaker the holistic view gets. In the latest survey [3], 68% of companies identified data silos as their primary problem, a seven percentage point increase over the previous year. Such fragmentation increases the chances of hidden dependencies and makes impact assessment more challenging. It leaves the pipeline open to cascading failures.

Even when technical relationships are reconstructed, business semantics are often missing: fields are treated merely as byte sequences rather than as metrics involved in reporting and risk management. Conservative metadata approaches hinder context propagation: the NewVantage survey in Harvard Business Review revealed that only 39% of respondents consider their organizations capable of managing data as an asset, and only 24% describe their company as data-driven, as shown in Figure 1 [4].

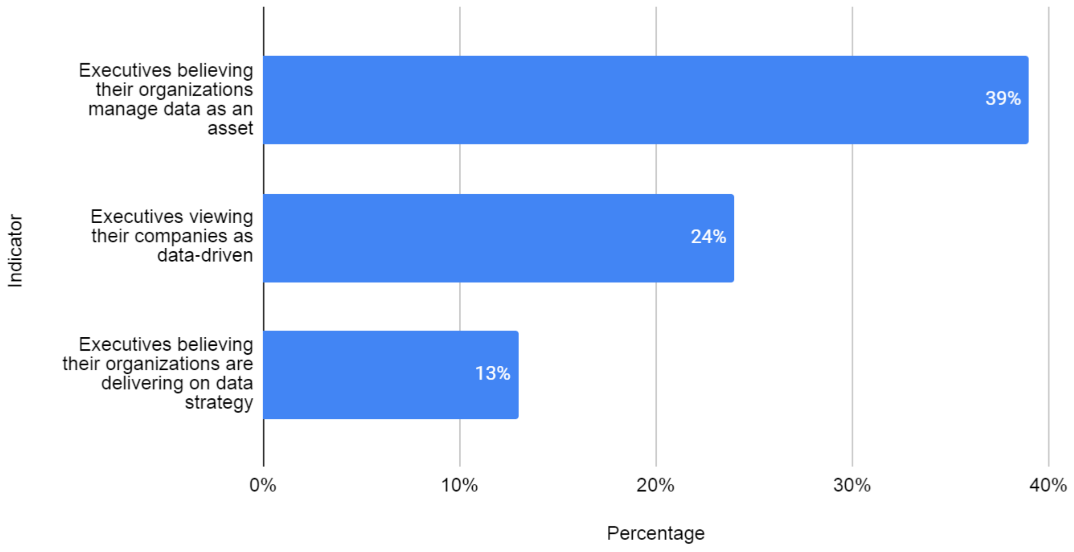

Fig. 1: Executive Perceptions of Organizational Data Management [4]

Without domain-language entity descriptions, even a full technical lineage fails to explain why a metric holds a specific value and which business unit would incur losses if it were distorted. Finally, dependency maps become outdated more quickly than new script versions are released, as each addition of a column or redistribution of tasks requires manual adjustments to the graph. According to the TDWI Monte Carlo–based survey, specialists spend approximately 40% of their work time on identifying and resolving data-quality issues, effectively devoting two days per week to remediation rather than platform development [5]. This lag between code changes and lineage updates causes organizations to act retrospectively, resulting in the loss of both resources and user trust.

A graph database treats data as a network of nodes and edges, thus any pipeline object—source file, intermediate table, transformation script, or report—becomes a vertex, and the flow or dependency between them is recorded as a directed edge. This topology mirrors the actual logic of data movement, relieving engineers of the need to reconstruct relationships via external joins and thereby restoring the cohesive provenance picture [6]. Storing relationships as first-class entities radically accelerates impact analysis: instead of cascading table intersections, the system performs a direct graph traversal, and even multi-step queries involving ten or more hops complete in milliseconds, which is crucial for interactive root-cause investigations and impact assessments before deployment [7]. Moreover, the schema remains flexible: new node types or attributes can be added without incurring costly migrations, and historical edges are retained in the same store, allowing for multiple lineage versions to be concurrently maintained. Such elasticity is inherent to the property-graph format, which allows for partial or fully optional schemas, thereby lowering the barrier to model evolution as business and regulatory demands grow, as illustrated in Figure 2.

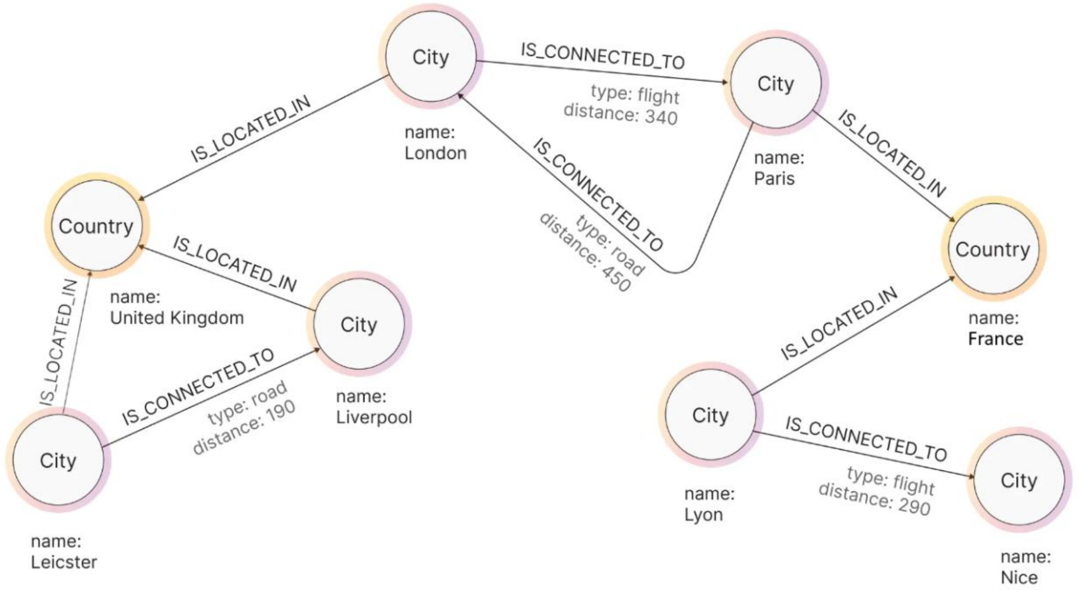

Fig. 2: The property graph model [7]

A traceability AI agent is an autonomous service that, in real-time, collects telemetry from orchestrators and version-control systems, interprets changes using language models, and immediately records the results in a graph database, thereby eliminating the lag between an event and its reflection in the lineage. The industry is already shifting toward such solutions: 80% of surveyed companies continue to consider data integration the main obstacle to scaling AI, which underscores the need for a unified metadata assembly mechanism [8].

The basic processing chain in such an agent consists of four interconnecting components. First, the event-subscription module captures standardized messages, compatible, for example, with the OpenLineage specification, and places them in a streaming queue, preserving a reference to the execution context. Next, the linguistic-analysis block applies large language models to SQL, Python, or Spark code to extract tables, fields, and operations; the practical few-shot prompting methodology for this task is described in [9], where a significant improvement in extraction accuracy compared with classical parsers is demonstrated, as shown in Fig. 3.

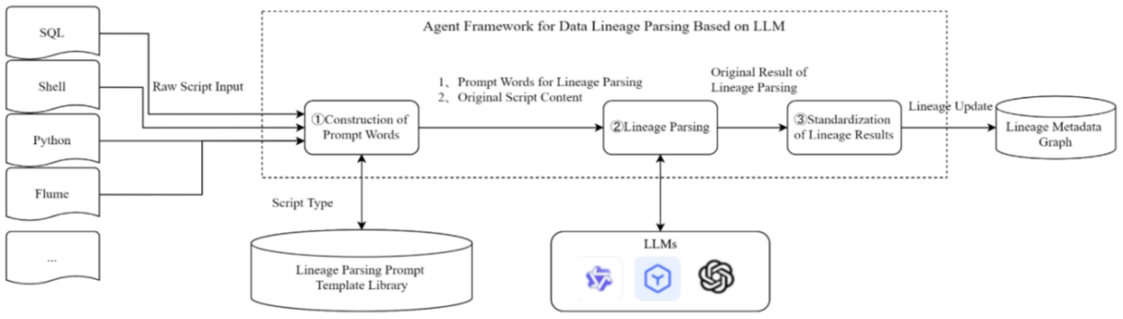
SQL
Shell
Python
Flume
...
Raw Script Input
Agent Framework for Data Lineage Parsing Based on LLM
1、Prompt Words for Lineage Parsing
2、Original Script Content
Original Result of Lineage Parsing
①Construction of Prompt Words
②Lineage Parsing
③Standardization of Lineage Results
Lineage Update
Lineage Metadata Graph
Script Type
Lineage Parsing Prompt Template Library
LLMs

Fig. 3: The overall process of data lineage parsing based on LLM [9]

The outputs pass through a machine-inference layer that validates new relationships and, if necessary, corrects them based on signals from the Kafka schema-change stream. Afterward, the write driver adds vertices and edges to the graph, supporting multiple versions of dependencies within a single model.

The agent's intelligent functions are built on three key algorithm classes. First, large language models trained on Text-to-SQL tasks are capable of interpreting even atypical query dialects, thus bridging the gap between technical code and business terminology [10]. Second, graph neural networks utilize the lineage topology to predict missing edges, enabling the identification of hidden dependencies before they lead to failures. A review of dynamic link-prediction methods reveals a steady increase in the accuracy of these models across various graph types [11]. Third, reinforcement-learning mechanisms introduce a feedback loop: the agent receives a reward when its automated inference matches manual validation or improves data quality, as has already been shown in a human-in-the-loop prototype for monitoring the quality of a physical experiment, as well as in corporate experiments at Databricks where the combination of RL and synthetic datasets improved model robustness without cleansing the source data [12]. This hybrid approach creates a self-tuning system that continuously refines its understanding of data flows, thereby transforming the infamous black-box graphs into a transparent, adaptive map of information provenance.

Workflow orchestrators, version-control systems, and monitoring services generate unified OpenLineage events, which describe the dataset, its state, the execution identifier, and the parameters used for processing. OpenLineage support is already built into Airflow via the official provider, so at each task run, metadata is generated automatically and forwarded to an external transport without the need for additional code [13].

To prevent message loss and simplify horizontal scaling, events are passed through a Kafka queue that guarantees at least once delivery and, when transactional mode is enabled, exactly once delivery, which is especially important for constructing continuous and consistent lineage under high load [14].

The AI agent subscribes to the relevant topics, enriches the events with business semantics, and writes them to the Neo4j graph database. The UBS practical case demonstrates that this model provides instant answers to multi-hop lineage queries and facilitates regulatory compliance, a feature not possible with relational stores that require numerous joins [15]. Over the graph, an API layer is implemented, offering REST and GraphQL endpoints that enable both services and analytical tools to request impact paths or create interactive visualizations.

During the middle of the workday, from-end-to-end monitoring detects an abnormal jump in the income metric in the monthly report, which deviates from the baseline confidence interval. The event is sent to the message queue, and immediately, the lineage agent retrieves the matching execution context, which stores the task identifier, code version, and run parameters.

Then, the agent dynamically queries the graph database, starting at the problematic metric, by traversing the produces and depends_on edges against the flow of the pipeline. The traversal takes fractions of a second and leads to the specific sales-aggregation script that was modified by the last nightly deployment. As file hashes are stored in the graph vertices, the agent compares the new and previous versions, identifies the modified filter condition, automatically determines that it excluded the entire returns category, and marks this node as the probable root cause.

The visualization interface displays the path from the report to the source table, highlights the responsible commit reference, and offers two options: roll back the changes to the last stable revision in the repository or apply a corrective patch that adds a returns-condition to the current logic. The user decides to roll back, which triggers a job in the orchestrator by the agent. It re-computes the affected stages and once done successfully, marks the incident as resolved and also logs feedback for later use in model training.

A single-event contract initiates dependable release. Take OpenLineage, for example. The specification outlines a canonical message format, eliminating the need for manual conversions. When orchestrators, version controls, and monitors all tools dispatch events in a standardized way, the agent receives a harmonized picture of each run, allowing developers to validate metadata with static tests before merging code into the main branch. This unification raises the bar on how predictable model behavior can be , not to mention lowering risks of leaving out bits from the graph, which is key for fast failure localization.

Another level of reliability is formed by an access-control system in which permissions for vertices and edges are clearly defined. Analysts may look at aggregated metrics without column sensitivity being revealed, while engineers freely explore technical dependency paths with no business content, such policy granularity is easily expressed by graph database labels and automated audits that check current rules against a canonical description in the infrastructure repository enforce it.

For reproducibility, the graph conserves all modifications in time, such that for every change, a new version of the graph is created with its history preserved through historical edges. This enables both states of the pipeline to be compared, report reproduction as of some past point in time, and validation that machine-learning models were trained on the same dataset used in the original experiment. The temporal axis streamlines investigations by indicating precisely when a critical dependency appeared and which commit introduced it.

Finally, the agent achieves maximal accuracy only through a robust feedback loop. The interface must allow an engineer to mark any automatically created edge as correct or erroneous, after which the model is retrained in the background on these annotations. This iterative cycle progressively reduces the need for manual corrections and increases the agent’s prediction confidence. To prevent concept drift, it is advisable to schedule regular quality assessments, in which experts and the resulting metrics verify that samples of new dependencies are automatically published to the monitoring system.

The integration of the agent with the graph database drastically shortens the incident-response cycle, as each pipeline stage is represented as a vertex and all transitions are recorded as edges, enabling the system to immediately construct the path from an anomalous metric to a specific commit or launch parameter. The engineer receives a ready-made causal map before even opening log files, so instead of hours of manual investigation, only a decision to recompute or roll back remains.

Complete and continuous lineage transforms regulatory requirements into verifiable rules. The graph stores version histories, transformation parameters, and references to source data, allowing an auditor to reconstruct any report in its original context by issuing a single query. As access to vertices and edges is governed by policies, confidential information remains protected, and the organization demonstrates process transparency, reducing the risk of fines and expediting audits.

When users see that every metric is linked to source tables and code, trust in analytical products increases. Reports cease to feel like black boxes, since any figure can be instantly traced to its provenance and change history. Such transparency enhances business-unit engagement, simplifies metric alignment, and makes communication between analysts and stakeholders more predictable.

Moreover, the graph model, combined with the agent's machine inference, enables the evaluation of the change's impact before deployment. The system identifies all objects that will be affected by a new script or migration and computes potential deviations of key metrics. The team receives a quantitative risk assessment, allowing them to select an optimal release window or apply adjustments in advance, thereby reducing the likelihood of incidents and improving platform stability.

Intelligent lineage agents integrated with graph databases deliver a single, adaptable, and current view of data origin. This not only speeds up incident discovery and resolution but also brings business value to the entire front. Versioning, along with old ties in the property graph, eliminates manual moves and reduces regulatory-compliance risk. The loop back through checking and model retraining steadily minimizes the need for manual fixes. Firms gain a steady way to assess the effects of changes before rollout, which boosts confidence in analytics and supports the growth of green platforms.

Conclusion

It proves the concept for the effectiveness of integration between autonomous AI-based lineage agents and graph databases toward building the possibility of holding in place a complete and up-to-date data provenance picture over the whole ETL pipeline, collecting metadata, and processing it, reducing dramatically the time needed to discover an incident by traversing directly a property graph instead of making several join operations. The flexible property-graph schema enables model evolution without costly migrations, preserving historical dependency versions and business semantics per node—thus, high-quality analytics, potential, and easy support for regulatory compliance are achievable. Support OpenLineage with Kafka queues guarantees consistency and completeness of events happening in the system. Exposure of this lineage graph through REST and GraphQL API layers is included as part of the proposed solution. A strong feedback loop, with real-time validation that reduces manual corrections required today, but will even mitigate them going forward by retraining itself on expert annotations. Robust validation before deployment equates to risk-aware release planning, building trust in analytical products, and enabling sustainable platform scaling.

References

[1]. G. Mathur, "Data science vs. machine learning: What's the Difference?" IBM, Aug. 21, 2024. https://www.ibm.com/think/topics/data-science-vs-machine-learning (accessed Jun. 26, 2025).

[2]. Gartner, "Data quality: Why it matters and how to achieve it," Gartner, 2024. https://www.gartner.com/en/data-analytics/topics/data-quality (accessed Jun. 27, 2025).

[3]. M. Knight, "Data Strategy Trends in 2025: From Silos to Unified Enterprise Value - DATAVERSITY," Dataversity, Dec. 03, 2024. https://www.dataversity.net/data-strategy-trends-in-2025-from-silos-to-unified-enterprise-value/ (accessed Jun. 28, 2025).

[4]. P. Sainam, S. Auh, R. Ettenson, and Y. S. Jung, "How Well Does Your Company Use Analytics?" Harvard Business Review, Jul. 27, 2022. https://hbr.org/2022/07/how-well-does-your-company-use-analytics (accessed Jun. 29, 2025).

[5]. "Data Engineers Spend Two Days Per Week Firefighting Bad Data, Survey Says," TDWI, 2022. https://tdwi.org/articles/2022/08/09/data-engineers-and-bad-data-survey.aspx (accessed Jul. 01, 2025).

[6]. A. P. Narayanasamy and Y. Chen, "The Rise of Graph Databases: An Extensive Exploration of Applications and Potential Challenges," Journal of AI Analytics and Applications, vol. 2, no. 2, pp. 1–9, Dec. 2024, doi: https://doi.org/10.63646/ogso5407.

[7]. Memgraph, "Graph data model," Memgraph, 2024. https://memgraph.com/docs/data-modeling/graph-data-model (accessed Jul. 03, 2025).

[8]. G. Sevilla, “AI agents go mainstream, but data integration remains a major hurdle,” Emarketer, Jan. 29, 2025. https://www.emarketer.com/content/ai-agents-go-mainstream--data-integration-remains-major-hurdle (accessed Jul. 03, 2025).

[9]. Z. Li, W. Guo, Y. Gao, D. Yang, and L. Kang, “A Large Language Model-Based Approach for Data Lineage Parsing,” Electronics, vol. 14, no. 9, p. 1762, Apr. 2025, doi: https://doi.org/10.3390/electronics14091762.

[10]. L. Shi, “A Survey on Employing Large Language Models for Text-to-SQL Tasks,” Arxiv, 2024. https://arxiv.org/html/2407.15186v5 (accessed Jul. 05, 2025).

[11]. N. A. Holagh and Z. Kobti, “Survey of Graph Neural Network Methods for Dynamic Link Prediction,” Procedia Computer Science, vol. 257, pp. 436–443, Jan. 2025, doi: https://doi.org/10.1016/j.procs.2025.03.057.

[12]. O. J. Parra et al., “Human-in-the-loop Reinforcement Learning for Data Quality Monitoring in Particle Physics Experiments,” Arxiv, May 2024, doi: https://doi.org/10.48550/arxiv.2405.15508.

[13]. “Documentation,” OpenLineage, 2025. https://openlineage.io/docs/ (accessed Jul. 10, 2025).

[14]. “Documentation,” Apache Kafka. https://kafka.apache.org/documentation/ (accessed Jul. 12, 2025).

[15]. Neo4j, “Data Lineage Tool Improves Risk Management, Drives Compliance,” Neo4j. https://go.neo4j.com/rs/710-RRC-335/images/Neo4j-case-study-UBS-EN-US.pdf (accessed Jul. 14, 2025).