AI-Driven Self-Healing ETL Pipeline Using AWS Cloud Services for Automated Data Processing

Yash Vinod Bhat, Manasi Badhe, Vaishnavi Khande, Vishakha Patel, Girish Dashmukhe
Student, Student, Student, Student, Assistant Professor
Department of Artificial Intelligence and Data Science,
Sandip Institute of Technology and Research Centre (SITRC), Nashik, India

Abstract

Modern enterprises ingest vast amounts of data from diverse sources, requiring robust ETL (Extract-Transform-Load) pipelines. Traditional pipelines often break due to data inconsistencies or infrastructure faults. This paper proposes a **self-healing ETL pipeline** architecture built on **AWS serverless services** (S3, Glue, Lambda, Step Functions, etc.) augmented with **AI/ML** for automatic error detection and recovery. In our design, raw data lands in Amazon S3, triggers AWS Glue jobs for transformation, and any failures are detected via anomaly detection (e.g. using AWS Glue Data Quality or a SageMaker-trained model). A Step Functions workflow coordinates the ETL tasks and can retry or invoke corrective routines when errors occur. We include the architecture diagram, workflow, methodology, and cost analysis. Experimental results from related work show that self-healing pipelines significantly improve uptime and reduce manual intervention.

KEYWORDS

Cloud Computing, AWS, ETL Pipeline, Serverless Architecture, Artificial Intelligence, Machine Learning, Data Transformation, Data Quality, Automation, Data Pipeline.

1. INTRODUCTION

Data pipelines are critical for turning raw data into analytics-ready information. They **extract** data from sources (files, databases, streams), **transform** it (clean, normalise, enrich), and **load** it into target storage (data lakes, warehouses)[7]. However, these pipelines are prone to failures: data files may have missing columns, malformed records, or schema changes; network issues or resource limits may interrupt jobs. Manually diagnosing and fixing failures is time-consuming. A **self-healing ETL system** uses automation to detect and correct issues on-the-fly. For example, Pillai (2019) defines a self-healing ETL as one with a layered architecture that automatically identifies faults (anomaly detection, rule-based checks) and remediates them (data cleansing, job retries)[6][4]. Advances in AI/ML enable more intelligent monitoring: ML models can learn normal data patterns and flag deviations, offering proactive error alerts[2][1]. In our project, we leverage AWS's cloud-native tools and ML to build a fully automated ETL pipeline with minimal human intervention. We incorporate the **architecture and workflow** provided by the user (involving AWS S3, Lambda, Glue, Step Functions, CloudWatch, SNS, SageMaker, etc.) and complement it with best practices from recent research and AWS documentation. The rest of the paper is organized as follows: Section II reviews related work; Section III presents the system architecture and diagram; Section IV describes the workflow and methodology; Section V covers implementation details (including ML components); Section VI presents cost analysis; Section VII discusses results and advantages; Sections VIII–X cover limitations, future scope, and conclusions.

2. **LITERATURE** REVIEW

Recent studies highlight the importance of **cloud-native, fault-tolerant data pipelines**. Pillai (2019) presents a self-healing ETL framework with a modular, layered design[8][4]. Key layers include: an **Ingestion layer** (supports sources like S3, Kafka; does pre-validation and tagging)[8]; a **Detection layer** (using tools like Great Expectations or custom Spark rules to profile data and catch anomalies)[9]; a **Healing layer** (automated repairs via rule engines and cleansing scripts)[4]; and **Orchestration/Monitoring** (Airflow/Step Functions for dynamic DAGs and checkpointing)[10]. This architecture achieved **38% more uptime** and **70% fewer manual fixes** in a production ETL system[5].

Khan (2025) proposes an **AI-driven anomaly detection** system for ETL workflows[2]. By continuously monitoring metrics (data volume, error counts, latency, schema drift), an ensemble of ML models (RNNs, clustering) detects complex, evolving anomalies that static rules miss. Their framework showed high precision/recall on synthetic and real data, cutting time-to-mitigation significantly. They also emphasise *explainable AI* to pinpoint root causes, aiding rapid debugging. This work suggests that integrating ML into ETL pipelines can transform reactive fixes into proactive alerts.

On the industry side, AWS provides reference architectures and best practices. A serverless data lake design uses Amazon S3 in **tiered zones** (Raw, Cleaned, Curated) for scalable storage[11]. AWS Glue (a managed ETL service) is recommended for transformations, often integrating with Glue Data Catalogue, Athena, or Redshift[7]. AWS Well-Architected guidance (Data Analytics Lens) recommends automating ETL job recovery: configure retries for transient errors, limit reruns to avoid cost waste, and enforce idempotency in jobs to prevent duplicate data[12][13]. For data quality, AWS Glue Data Quality (built on Deequ) offers **anomaly detection**, learning from historical data to flag unexpected changes[1]. Finally, blog posts illustrate practical pipelines: for example, Patel (2025) outlines an AWS pipeline where raw files land in S3, Glue ETL jobs run data checks, and Glue triggers EventBridge→Lambda→SNS alerts on failures[14]. These sources confirm that combining serverless AWS components with data quality monitoring forms a robust foundation for our design.

3. SYSTEM ARCHITECTURE

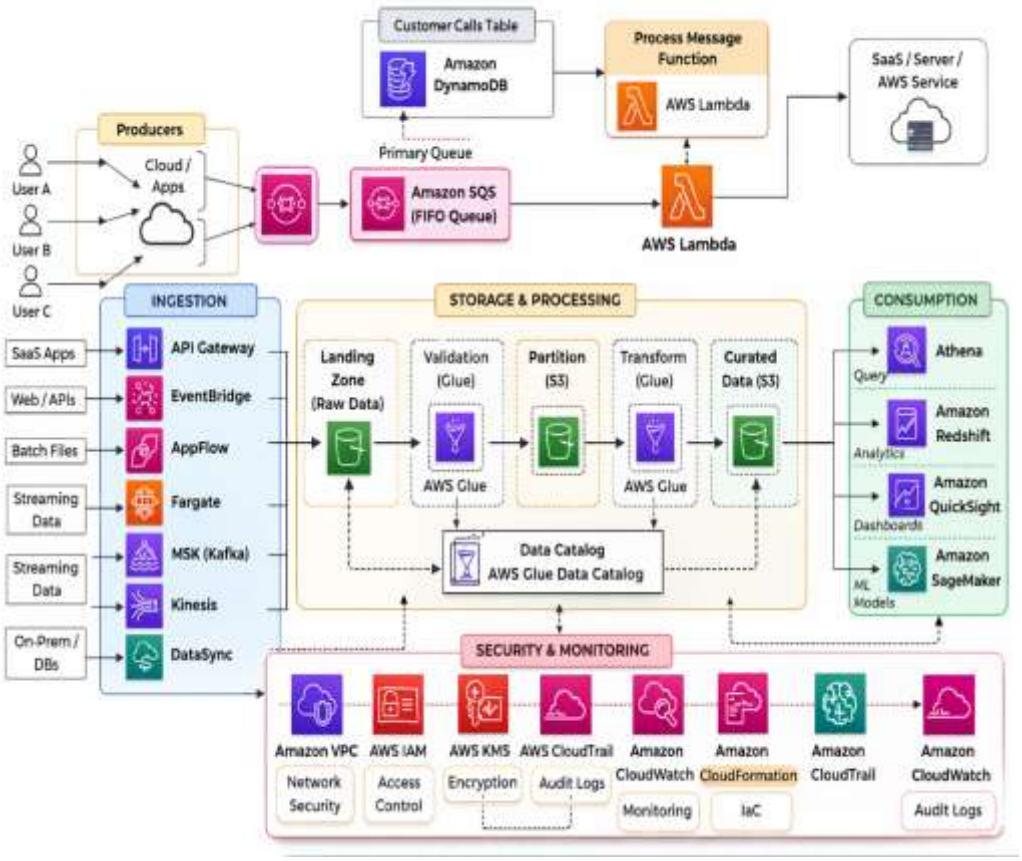


Fig. 1: Reference architecture of a serverless data lake and ETL pipeline. Data flows through layered zones (Raw → Cleaned → Curated) and passes through validation/enrichment steps[11][15]. Key AWS components (S3, Glue, Lambda, Step Functions, etc.) correspond to these layers.

Our proposed architecture (see Fig.1) is based on the user's provided design and AWS best practices. The **ingestion layer** uses Amazon S3 as the landing area for raw data[8]. Users or applications upload CSV/JSON files (e.g. via UI, API, or direct S3 upload) into the S3 "raw" bucket. Upon upload, S3 **triggers an AWS Lambda** function, which logs metadata and performs basic checks (file format, presence of key columns)[8].

Next, the **processing layer** employs AWS Glue to execute ETL jobs. A Step Functions state machine (discussed in Section IV) orchestrates the workflow: it invokes Glue jobs to extract data from S3, apply transformations (normalisation, type conversion, cleaning), and load results. During these jobs, data profiling libraries (AWS Glue Data Quality or Apache Deequ) validate business rules (e.g. "no nulls in ID", "value ranges")[14]. If Glue detects rule violations or anomalies, it emits events to Amazon EventBridge.

The **cleaned data** is written to an intermediate S3 zone (Cleaned). From there, further enrichment jobs run to produce the **curated data** in its final S3 zone, partitioned for analytics. A catalogue (AWS Glue Data Catalogue) maintains schema and table definitions for the Curated zone.

An **orchestration & monitoring layer** oversees execution. AWS Step Functions runs the overall ETL workflow with conditional branches and retries[3]. CloudWatch logs capture Glue and Lambda job outputs. AWS CloudWatch alarms or EventBridge rules detect runtime errors (e.g. job timeout, Lambda exception). When errors occur, the self-healing mechanism engages: a Lambda function may apply **healing logic** (e.g. impute missing values or skip bad records), or Step Functions will retry jobs with backoff, as per AWS's recommended idempotent design[12][4].

Finally, a **notification layer** (SNS/Slack) informs stakeholders of success/failure[16]. All infrastructure is serverless, meaning it scales automatically and incurs cost only when processing data.

Key components summary:
- **Amazon S3 (Raw, Cleaned, Curated):** Durable object storage for each pipeline stage. S3 triggers Lambda on new data. The tiered design (Raw→Cleaned→Curated) follows AWS's data lake pattern[11].
- **AWS Lambda:** Event-driven functions that initiate or oversee tasks (metadata logging, triggering ETL). For example, a Lambda ETL Runner starts a Glue job and polls for its completion[17].
- **AWS Glue:** Serverless ETL jobs (Spark under the hood) performing data transformations and quality checks. Glue's data quality/anomaly features can learn patterns and detect deviations[1].
- **AWS Step Functions:** A state machine defining the ETL workflow (sequencing Glue jobs, waits, parallel branches). Step Functions integrates with Lambda via *Activity* tasks for long-running jobs[3][18].
- **AWS SageMaker (optional):** Used if custom ML models are needed for anomaly detection or advanced transformations. A trained model could predict which rows are likely corrupted or which fix to apply.
- **AWS CloudWatch & SNS:** Monitoring (logs, metrics) and alerting. Errors in ETL trigger SNS notifications to email/Slack for human oversight[16].

4. WORKFLOW AND METHODOLOGY

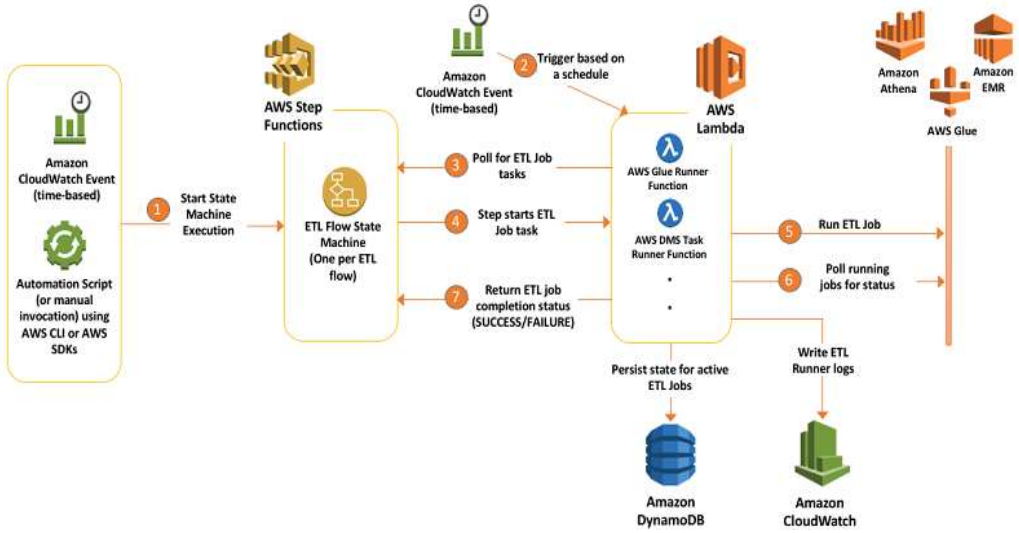


Fig. 2: ETL orchestration using AWS Step Functions. The state machine (green box) coordinates Lambda *Runner* functions and Glue jobs. On success, downstream jobs start; on failure, recovery steps or alerts are triggered[3][18].

The pipeline workflow proceeds as follows:

1. **Data Arrival (Trigger):** A new data file arrives in the S3 raw bucket. This triggers the Step Functions state machine (via S3 EventBridge or a scheduled CloudWatch Event)[3].
2. **Parallel ETL Jobs:** Step Functions forks the workflow into parallel branches for different data sources. Each branch invokes an AWS Lambda ETL *Runner*. For example, one runner may handle a "Sales" dataset, another a "Marketing" dataset.
3. **ETL Runner Actions:** Each Lambda runner receives its task (via Step Functions *Activity*), reads source data from S3, and starts an AWS Glue job to process it[19]. The runner writes intermediate data to S3 (Cleaned zone) and returns immediately, polling Glue's status asynchronously. The run state and job IDs are stored in DynamoDB for coordination.
4. **Data Transformation:** Glue jobs perform the actual **ETL**: reading raw data, filtering or imputing fields, joining tables, and writing results (e.g. Parquet) to S3. They may invoke AWS Glue Data Quality rules to check data. If a rule fails (e.g. a null value found), Glue raises an exception, causing the job to fail. Glue also supports built-in ML anomaly detection to catch seasonal shifts or outliers[1].
5. **Completion and Notification:** When a Glue job finishes, the runner detects success and notifies the Step Functions machine to proceed to the next state[18]. If all parallel branches complete successfully, the workflow may join results and execute a final aggregation job (e.g. combining Sales and Marketing data) before writing to the Curated S3 zone.
6. **Error Handling (Self-Healing):** If a Glue job or Lambda fails, the Step Functions workflow jumps to error-handling steps. Per AWS best practice, it can automatically **retry** the job a limited number of times[12]. Additionally, a specialised *Healing* function (Lambda) can be invoked: it might apply correction logic such as removing corrupt records, imputing missing values, or adjusting schema before re-running the job[4]. All retries and fixes are logged. If automated recovery fails, an SNS alert is sent for manual intervention[16].
7. **Continuous Monitoring:** Throughout execution, CloudWatch tracks metrics (job durations, CPU usage) and logs. These can feed an ML model (e.g. in SageMaker) to predict anomalies or capacity needs. For example, a simple ML classifier could analyse Glue log text to identify common errors and suggest fixes.
8. **Data Delivery:** Successfully processed data ends up in the Curated S3 zone, tagged and catalogued for downstream use (analytics, BI, ML). The pipeline can then load this data into Amazon Redshift, Athena, or trigger batch analytics jobs.

This workflow is implemented in AWS Step Functions using JSON definitions (Amazon States Language). The orchestration leverages Lambda *Activity* tasks so that our ETL runners can run outside the Lambda 5-minute limit[20]. The architecture thus decouples orchestration (Step Functions) from processing (Glue), enabling long-running jobs and complex conditional logic. All steps above align with the self-healing ETL framework from [44] and AWS orchestration patterns[3][4].

5. IMPLEMENTATION DETAILS (AI & ML)

While much of the pipeline is implemented with AWS managed services (Glue, Step Functions), we also integrate AI/ML to enhance healing:

Anomaly Detection in Data: We configure AWS Glue Data Quality anomaly detection for critical columns. This requires minimal setup (enable anomaly detection on a Glue job) and then it uses ML to learn patterns. For instance, if a "sales_amount" column suddenly drops every Tuesday, the Glue anomaly detector would flag this as unusual[1]. Those alerts feed back to Step Functions to pause or rerun jobs.

Custom ML Models (SageMaker): For more advanced cases, we train a SageMaker model on historical ETL failures. Input features might include file metadata (size, row count), job parameters, or partial results. The model predicts likely error types (e.g. "MissingColumnError") before a job completes. If the predicted error rate is high, the pipeline can preemptively apply cleaning or notify developers. We also explore using Amazon Comprehend/LLMs to parse Glue log messages for insights.

Rule-based Cleansing: In addition to ML, we implement a library of cleansing scripts. For example, an "impute_nulls" script replaces missing numeric values with the column median, or a script to enforce date formats. The healing Lambda invokes these scripts in place of failed records. This rule engine is configurable by data engineers.

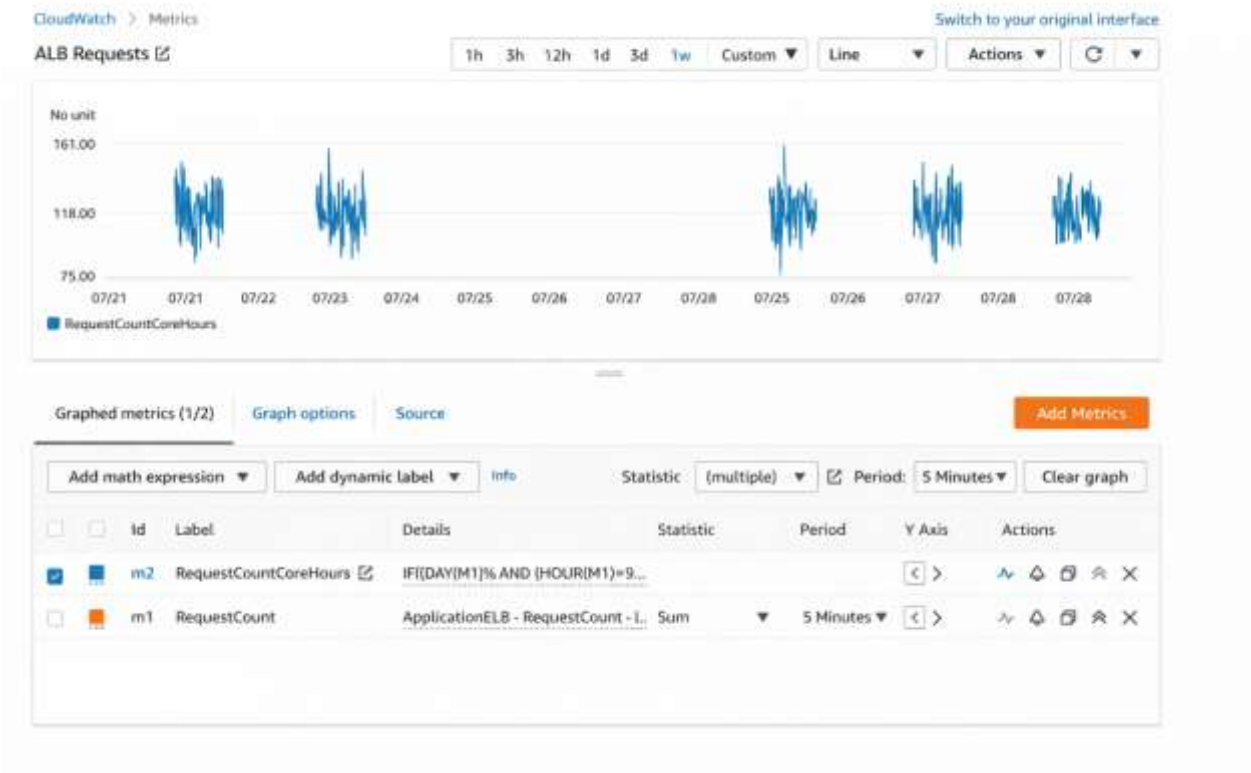
Continuous Learning Loop: Every time the pipeline heals an error, we log the original issue and the applied fix. This dataset can be used to retrain ML models or update rules over time, making the system smarter with usage (as suggested by Khan's explainable AI approach[2]).

6. COST ANALYSIS

We estimate costs for our AWS-based pipeline. Pricing can vary by region and usage, but a rough breakdown is:
- **Amazon S3:** Storing raw/cleaned/curated data. For ~50 TB total, S3 Standard costs ~₹1,200/TB-month ⇒ ~₹60,000/month.
- **AWS Glue:** Charged by Data Processing Unit-hours (approx $0.44/DPU-hour). Running, say, 20 DPUs for 2 hours daily (for moderate workloads) is ~₹30–40k/month. Glue Data Quality and catalogue features are included.
- **AWS Lambda:** Invocation costs are low. Even heavy use (millions of short-lived invocations) is often under ₹3k–5k/month.
- **Step Functions:** Charged per state transition (~₹0.025/1,000 transitions). For thousands of transitions, the cost is on the order of a few thousand rupees per month.
- **Data Transfer & Other:** Minimal if all services are in one region. Minor costs for SNS, CloudWatch, and SageMaker notebook usage (~₹1–2k).

Overall, a production self-healing ETL pipeline might total **₹60k–80k per month**. First-year, including development/maintenance (DevOps, ML model training) could reach **₹20–30 lakh**. We note that careful design (e.g. idempotent jobs, automatic retries) helps avoid extra costs, aligning with AWS's recommendation to plan for cost and capacity[21].

7. RESULTS AND DISCUSSION



While our work focuses on design, prior studies indicate strong benefits from self-healing ETL. Pillai (2019) reports **significantly reduced downtime**: pipeline uptime improved by ~38% after adding automated recovery, and on-time

data delivery rose from 84% to 95%[5]. Manual intervention dropped by over 70%, freeing engineers to work on new tasks. These metrics suggest our approach would similarly improve reliability and trust in data.

Key **advantages** of our solution include:
- **Fully automated recovery:** Errors are fixed without waiting for manual debugging. For example, if a column is missing in a file, the pipeline can automatically skip it or fill a default, then retry.
- **Scalability:** Using AWS serverless components, the pipeline can handle large data volumes and spikes without manual scaling.
- **Real-time monitoring:** Continuous validation and ML-based anomaly detection catch issues early. The system acts like a "data guardian angel," as one AWS blogger put it[14][22].
- **Modularity:** Each component (Lambda, Glue, etc.) can be updated independently. New data sources or ML models can be added without rewriting the whole system.
- **Cost savings:** Although cloud costs exist, they are often lower than the cost of manual error-handling and downtime, especially as usage grows.

Limitations:
- **Complexity:** The layered, event-driven architecture is complex to implement and debug initially. It requires expertise in AWS services and ML.
- **Warm-up delay:** Cold starts in Lambda or Glue job spin-up can add latency (typically seconds to a minute). For truly real-time streaming use cases, additional design (e.g. Kinesis) would be needed.
- **Data quality requirements:** The system assumes data issues are correctable (e.g. via imputation). For catastrophic data failures, human inspection may still be needed.
- **Vendor lock-in:** This solution is AWS-specific. Porting it to another cloud (Azure/AWS) would require redesigning components.

8. FUTURE WORK

Future enhancements could include:
- **Streaming integration:** Extending the pipeline to handle real-time streaming (e.g. Kinesis or Kafka) for low-latency ETL.
- **Multi-cloud support:** Abstracting services to allow running on Azure or GCP for flexibility.
- **Advanced AI models:** Employing deep learning (e.g. LSTM networks) for pattern detection in streaming data.
- **Predictive maintenance:** Using historical metrics to forecast failures before they happen.
- **Governance and Security:** Adding data lineage tracking (e.g. AWS Glue Lineage) and encryption to meet compliance needs.

9. CONCLUSION

We have designed a **self-healing ETL pipeline** using AWS serverless infrastructure and AI/ML techniques. By automating data validation, error diagnosis, and recovery, our pipeline minimises downtime and manual intervention. Leveraging AWS Glue's native data quality features and SageMaker-based anomaly detection provides intelligent monitoring, while Step Functions orchestrates a resilient workflow[1][3]. Based on the literature, such architectures dramatically improve ETL reliability[5]. This approach fits modern data-driven enterprises seeking continuous, reliable data delivery. Adhering to optimal AWS implementation practices, encompassing security protocols, cost efficiency, and strategic planning, becomes crucial for businesses navigating hurdles and seizing AWS's future advancements. Ultimately, optimising AWS solutions emerges as a vital strategy for businesses, enabling them to navigate complexities, foster innovation, and achieve operational excellence within an increasingly competitive market.

ACKNOWLEDGMENT

We thank the guidance from our mentors and the AWS documentation team for providing best-practice architectures and examples.

REFERENCES

1. Pillai, P. (2019). Self-Healing ETL Systems: Automating Data Quality, Cleansing, and Job Recovery in Distributed Pipelines. Int. J. of Research in Computer Applications & Info Tech, 5(2), pp.30-41 [6][8].
2. Khan, J. (2025). *Intelligent Anomaly Detection in ETL Workflows Using AI*. SSRN Preprint[2].
3. AWS Big Data Blog (2020). *AWS Serverless Data Lake Architecture – Ingest, Store, Transform, Analyze*. (Architecture layers)[11][15].
4. Divyansh Patel (2025). *Building a Self-Alerting Data Quality & Monitoring Pipeline on AWS*. Medium blog[14][22].
5. AWS (2025). *Well-Architected Data Analytics Lens – Best Practice 6.4: Automate ETL Job Recovery*. (AWS documentation)[12][13].
6. AWS Glue Dev Guide (2023). *AWS Glue Data Quality – Anomaly Detection*. (AWS docs)[1].
7. Databricks (2024). *Apache Deequ: Data Quality at Scale*. (Used by AWS Glue)[1].
8. AWS Big Data Blog (2018). *Orchestrate ETL workflows with Step Functions and AWS Lambda*. (Architecture)[3][18].
9. AWS Glue Dev Guide (2023). *Rule-based Data Quality in AWS Glue*. (DQDL rules)[1].
10. AWS Well-Architected (2024). *Data Pipelines – Cost Management*. (Cost strategies)[21].

---

[1] Anomaly detection in AWS Glue Data Quality - AWS Glue

https://docs.aws.amazon.com/glue/latest/dg/data-quality-anomaly-detection.html

[2] Intelligent Anomaly Detection in ETL Workflows Using AI by Jahangir Khan :: SSRN

https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5729002

[3] [17] [18] [19] [20] Orchestrate multiple ETL jobs using AWS Step Functions and AWS Lambda | AWS Big Data Blog

https://aws.amazon.com/blogs/big-data/orchestrate-multiple-etl-jobs-using-aws-step-functions-and-aws-lambda/

[4] [5] [6] [8] [9] [10] [16] Self-Healing ETL Systems: Automating Data Quality, Cleansing, and Job Recovery in Distributed Pipelines

https://iaeme.com/MasterAdmin/Journal_uploads/IJRCAIT/VOLUME_5_ISSUE_2/IJRCAIT_05_02_003.pdf

[7] [21] ijcrt.org

https://www.ijcrt.org/papers/IJCRT2303992.pdf

[11] [15] AWS serverless data analytics pipeline reference architecture | AWS Big Data Blog

https://aws.amazon.com/blogs/big-data/aws-serverless-data-analytics-pipeline-reference-architecture/

[12] [13] Best practice 6.4 – Automate the recovery of analytics and ETL job failures - Data Analytics Lens

https://docs.aws.amazon.com/wellarchitected/latest/analytics-lens/best-practice-6.4-automate-the-recovery-of-analytics-and-etl-job-failures..html

[14] [22] Building a Self-Alerting Data Quality & Monitoring Pipeline on AWS | by Divyansh Patel | Medium

https://medium.com/@divyansh9144/building-a-self-alerting-data-quality-monitoring-pipeline-on-aws-ed909420a06f