International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

A Secure and Automated Framework for GCP-Based D0/D1 Data Processing, Ingestion, and SQL Generation

Tanaya Kate[1], Abhishek Singh[2]

[1,2]Departmet of Technology, SPPU

Abstract

This paper presents a comprehensive analysis of a Streamlit-based D0→D1 automation system built on Google Cloud Platform (GCP) components, including Cloud Storage (GCS), BigQuery, and Vertex AI. The system supports ingesting Excel files from local or GCS sources, transforming them into structured BigQuery tables, exploring schemas, and generating stored procedures using a generative model (Gemini 3 pro). We analyze the implementation of the application , identify operational and security risks—such as hard-coded credentials, full-table in-memory downloads, PII exposure to LLM prompts, and unsafe SQL execution—and provide an improved, production-ready D0→D1 architecture. We conclude with a prioritized implementation plan, suggested code-level changes, and governance recommendations to make the pipeline secure, scalable, and auditable.

I. Introduction

Early-stage data ingestion and transformation are critical to enterprise analytics. A well-defined separation between the raw landing layer (D0) and the staged/validated layer (D1) enables auditability, repeatability, and data quality controls. This paper documents a concrete implementation of a D0→D1 automation application written in Streamlit and deconstructs its architecture, behavior, and risks. The goal is to present a rigorous, actionable blueprint that upgrades the current implementation to a production-grade, secure pipeline while preserving developer productivity and LLM-assisted automation.

The target audience includes data engineers, platform architects, compliance officers, and practitioners interested in operationalizing LLM-assisted SQL generation for data warehousing workflows.

II. Background and Literature Review

Layered data architectures such as the Medallion Architecture (Bronze/Silver/Gold) and modern lakehouse patterns emphasize clear boundaries between immutable raw storage and progressively refined datasets. D0 corresponds to the raw landing zone (immutable files/events), while D1 comprises cleaned and conformed datasets that are safe for analytical consumption. Prior work on reliable data pipelines highlights the importance of ingestion logs, schema enforcement, idempotent transformations, and observability (Marz & Warren, Kimball & Ross, Databricks Medallion pattern). Recently, the integration of large language models (LLMs) into data engineering workflows has enabled automated SQL generation and template creation.

Cloud-native implementations of these architectures, particularly within Google Cloud ecosystems, leverage services such as Cloud Storage and BigQuery to operationalize raw and staged layers. However, these approaches primarily address storage scalability and transformation reliability, without incorporating generative AI into structured pipeline transitions.

Recent research on LLM-to-SQL systems has focused on semantic translation accuracy and natural language interfaces for query generation. While these systems demonstrate high productivity gains, limited work addresses secure execution controls, governance constraints, and prompt-level data exposure risks in enterprise settings. Moreover, existing studies rarely formalize how generative models can be integrated into layered data architectures without compromising auditability or compliance.

III Methodology

This study follows a three-fold methodology:

Implementation Analysis

- Line-by-line and function-level review of code to map code regions to D0/D1 responsibilities, identify failure modes, and document side effects.

Threat & Risk Assessment

- Security and governance evaluation focusing on credentials, PII exposure to Vertex AI, SQL execution risks.

Design & Recommendation

- Propose architecture improvements, code-level mitigations, and an implementation plan prioritized by safety and operational impact.

IV. System Architecture and Implementation Details

4.1 Overview

The application is a Streamlit-based UI that orchestrates file ingestion from local uploads or a configured GCS bucket , parses Excel files into pandas DataFrames, and writes them into a BigQuery dataset called 'Test'. It also allows users to select existing BigQuery tables and columns, fetches sample data, and uses Vertex AI generative model to synthesize stored-procedure SQL which it can then execute against BigQuery.

4.2 Global Configuration and Initialization

Key global variables and initialization details are central to both operational behavior and security posture. The code uses a hard-coded service account JSON path (key_path), which is set into the environment variable GOOGLE_APPLICATION_CREDENTIALS so Google client libraries use Application Default Credentials (ADC). Vertex AI is initialized using the same project_id and location ('us-central1'), and a generative model object is created: GenerativeModel. Clients for BigQuery and GCS are created via functions decorated with @st.cache_resource to maintain cached connections across Streamlit reruns.

4.3 GCS Helpers

Functions list_gcs_files(bucket_name) and read_gcs_file(bucket_name, file_name) provide read-only listing and file download capabilities. Listing filters for '.xlsx' objects and returns a sentinel option. read_gcs_file downloads the blob as bytes and constructs a pandas DataFrame using pd.read_excel(io.BytesIO(data)). These functions do not currently implement size checks, exception handling for missing permissions or corrupt files, or streaming reads for large objects.

4.4 BigQuery Writer: upload_file(df, file_name)

upload_file converts a filename into a sanitized table name (removing spaces, hyphens, replacing dots with underscores, lowercasing). It checks for table existence with bigquery_client.get_table(table_ref) and presents Streamlit buttons to either append to an existing table or create a new table. Writing uses bigquery.Client.load_table_from_dataframe with a LoadJobConfig and job.result() to wait for completion. The function mixes UI (st.button) with persistence logic, which complicates testing and reuse.

4.5 Metadata Discovery and Column Fetching

The UI lists BigQuery tables using bigquery_client.list_tables(dataset_id) and fetches columns via bigquery_client.get_table(table_ref), returning a list of 'table.column' strings for UI selection. This helps users compose column lists, but the implementation loads full table data via client.query(...).to_dataframe() when building prompts, which can be highly inefficient and collision-prone if columns share names across tables.

4.6 Vertex AI SQL Generation

The core LLM-assisted feature is generate_sql_query(dict1, table_name, dict_table) based on your prompt instruction. The function constructs a prompt addressed to an 'expert in BigQuery SQL' and includes DataFrame heads for each referenced table directly in the prompt. The model is invoked (model.generate_content(prompt)) and the returned text is processed to extract SQL. The code currently trusts the LLM output and executes it via bigquery_client.query(sql_query) upon user button click.

V. Findings and Results

5.1 Mapping to D0/D1 Responsibilities

D0 (Raw/Landing) behaviors present:

- Ability to read and list raw Excel files from a GCS bucket and accept uploads from users.
- Original files can remain in GCS; however, the script does not enforce a D0 naming scheme (e.g., d0/{source}/{timestamp}) nor record a durable ingestion log.

D1 (Staged/Cleaned) behaviors present:

- Writing parsed DataFrames into BigQuery tables (create/append) with enforced column types through BigQuery's schema handling.
- SQL generation to create stored procedures for merges/updates represents an attempt to automate D1-to-D2 transitions.

VI. Discussion

The application embodies a common pattern seen in developer-led data tooling: fast iteration through direct access to cloud resources combined with minimal engineering controls. While this enables rapid prototyping and can significantly accelerate developer productivity, the lack of engineering safeguards poses operational, security, and compliance risks when the tool is used beyond trusted environments. Specifically, LLM integration magnifies risks because the model's output is not inherently trustworthy and the model call may leak sensitive content if prompts include raw data samples.

From an operational perspective, reliance on pandas for large data movement is unsustainable. Platform-grade systems should use scalable ingestion strategies (BigQuery load jobs from GCS, streaming inserts with chunking, or processing via distributed engines).

From a security and governance perspective, three key controls are essential before using such a tool in production: (1) secret management and least-privilege IAM, (2) data redaction or sanitized prompt construction, and (3) rigorous staging (dry-run, cost estimate, approval) for generated SQL.

VII. Experimental Evaluation

To validate the proposed architectural improvements, we conducted controlled experiments comparing the baseline implementation with the redesigned framework.

A .SQL EXECUTION SAFETY VALIDATION

We evaluated 20 generated SQL queries across merge and update scenarios.

Baseline behavior:

1. 3 queries included structurally unsafe patterns (missing WHERE clauses in updates).
2. No validation stage prevented execution.

Improved framework:

1. All queries underwent BigQuery dry-run validation.
2. Destructive operations (DROP, TRUNCATE) were blocked via rule-based filtering.
3. unsafe queries were flagged prior to execution.

This demonstrates improved operational safety through staged validation.

B. INGESTION SCALABILITY ASSESSMENT

The baseline system used pandas in-memory loading of Excel files prior to BigQuery ingestion. For datasets exceeding 50 MB, memory consumption increased significantly, and processing latency degraded.

The improved architecture leverages:

1. Direct GCS uploads
2. Bigquery load jobs

This reduces local memory dependency and improves scalability for larger ingestion workloads.

VIII. Novelty and Contributions

Although layered data architectures and generative AI–assisted SQL systems have been explored independently in prior studies, limited research has examined their secure operational integration within structured D0→D1 cloud data pipelines. In particular, governance-aware prompt engineering, execution-stage validation, and risk-controlled LLM deployment remain underrepresented in enterprise data engineering literature.

This study addresses this gap by proposing and analyzing a secure, cloud-native framework that integrates generative SQL synthesis into a D0→D1 automation workflow built on Google Cloud services. The principal contributions of this work are outlined below.

A. Structured Integration of Generative AI into D0→D1 Pipelines

Conventional layered architectures, including Medallion-based implementations promoted by Databricks, focus on progressive data refinement but do not explicitly incorporate generative AI within transformation stages. Similarly, cloud ingestion patterns emphasize scalability and reliability without formalizing AI-assisted SQL synthesis as a governed architectural component.

This research presents a structured integration of a generative model deployed via Vertex AI into the D0→D1 transition layer. The framework constrains model interaction through metadata-driven prompt

construction, schema-aware context injection, and execution-stage safeguards. This positions generative AI as a managed subsystem within the pipeline rather than an unregulated auxiliary tool.

B. Risk-Aware Operational Model for LLM-Assisted SQL Execution

Existing LLM-to-SQL systems primarily emphasize translation accuracy and query generation performance. In contrast, this work introduces a risk-aware operational model that explicitly addresses:

1. Potential exposure of sensitive data within LLM prompts
2. Execution of syntactically valid but operationally unsafe SQL statements
3. Credential management vulnerabilities in cloud-connected applications
4. Inefficient in-memory data handling during ingestion workflows

The proposed framework incorporates schema-only prompt strategies, sample-level data masking, dry-run validation, and role-based access control redesign. By embedding governance and validation mechanisms into the SQL execution lifecycle, the framework extends LLM-assisted automation into compliance-aligned enterprise contexts.

C. Formal Mapping Between Conceptual Architecture and Implementation

A methodological contribution of this study is the explicit mapping between conceptual D0/D1 layer definitions and their concrete realization within cloud services and application logic. Specifically:

- D0 responsibilities are operationalized through controlled file ingestion and immutable storage in Google Cloud Storage.
- D1 responsibilities are implemented through schema-enforced BigQuery load operations and transformation procedures.

This mapping provides a reproducible blueprint that bridges theoretical layered architecture principles with practical GCP-based deployment strategies.

D. Prototype-to-Production Hardening Framework

Enterprise data tools frequently originate as rapid prototypes (e.g., interactive applications developed using Streamlit). However, formal guidance on transitioning such tools to production-grade systems remains limited in academic discourse.

This study contributes a structured hardening roadmap encompassing:

- Secure credential management practices
- Separation of user interface and persistence layers
- Scalable ingestion patterns (e.g., GCS-to-BigQuery load jobs)
- Audit logging and observability mechanisms
- Controlled SQL execution workflows

This contribution addresses a practical but underexplored research area: systematically elevating developer-centric automation tools to enterprise-ready infrastructure.

E. Governance-Oriented Prompt Engineering Strategy

The study further introduces a governance-oriented prompt engineering methodology tailored for database mutation workflows. Instead of transmitting full dataset samples to the model, the framework emphasizes:

- Schema-level metadata exposure
- Limited, masked contextual sampling
- Explicit non-destructive query constraints
- Human-in-the-loop validation prior to execution

This structured approach mitigates data leakage risk while preserving the functional advantages of generative SQL synthesis. Such governance-driven prompt design remains insufficiently examined in current LLM-assisted data system literature.

Summary of Contributions-

In summary, this research contributes:

1. A structured framework for integrating generative AI into D0→D1 cloud data pipelines.
2. A risk-aware lifecycle model for LLM-assisted SQL generation and execution.
3. A formal alignment between layered data architecture concepts and concrete GCP implementation.
4. A production hardening roadmap for prototype-based cloud data applications.
5. A governance-driven prompt engineering methodology for secure database operations.

Collectively, these contributions extend existing layered data paradigms by incorporating generative AI within a controlled, auditable, and enterprise-oriented architectural framework.

IX. Conclusion

This paper provides a thorough analysis of a Streamlit-based D0→D1 automation application and offers a pragmatic, prioritized plan to upgrade it to a production-grade system. By implementing ingestion logging, sanitized prompt engineering, sampling, dry-run validation, and secure credential handling, teams can retain the productivity benefits of interactive LLM-assisted development while meeting enterprise requirements for security, scalability, and governance.

IX. References

1. Databricks. Medallion Architecture for Lakehouse Data Engineering. 2023.
2. Google Cloud. Designing data lakes with BigQuery and Cloud Storage. Google Cloud Architecture Center. 2024.
3. Kimball, R., & Ross, M. The Data Warehouse Toolkit. John Wiley & Sons.
4. Marz, N., & Warren, J. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. 2015.
5. Google Cloud Vertex AI Documentation. Generative Models (OpenAI, Gemini). 2024.
6. Streamlit Documentation. Best practices for secrets management and caching.