Analytics Copilot: Ask Questions About Your Data

CS 5542 Big Data Analytics & Applications - Project Proposal

Ben Blake (GenAI & Backend Lead)
Tina Nguyen (Data & Frontend Lead)

February 3, 2026

Abstract

This proposal outlines the design and implementation of an **Analytics Copilot**, an AI-powered system that democratizes data access by allowing non-technical users to query enterprise databases using natural language. Leveraging **Snowflake** as the core data engine and **Retrieval-Augmented Generation (RAG)** for schema context, the system translates plain English questions into executable SQL queries. By integrating recent advancements from NeurIPS 2025 such as schema abstraction and expert schema linking, our solution addresses the common "bottleneck" problem in data analytics where business users rely heavily on technical analysts. The project will feature a **Streamlit** interface for interaction, **Snowflake Cortex** for LLM inference, and a rigorous evaluation framework using synthetic "Golden Q&A" pairs.

Contents

1 Introduction

The rapid advancement of Large Language Models (LLMs) has revolutionized how humans interact with digital systems. However, in the domain of enterprise data analytics, a significant gap remains. While LLMs excel at generating code, they often struggle with the specific nuances of enterprise database schemas, leading to "hallucinations"—plausible but incorrect SQL queries. For non-technical business users, this unreliability is a critical barrier to adoption.

This project proposes an **Analytics Copilot**, a robust, Snowflake-native system designed to bridge this gap. By leveraging a Retrieval-Augmented Generation (RAG) architecture specialized for structural metadata, our system allows users to ask questions in plain English and receive accurate, executable SQL queries and visualizations. Unlike generic chatbots, our solution is grounded in the specific schema of the organization, utilizing advanced techniques like *schema linking* and *semantic abstraction* to ensure high fidelity.

We chose **Snowflake** as our platform because of its integrated AI capabilities (Cortex), ensuring that data never leaves the secure governance boundary of the warehouse. This proposal outlines the architectural design, implementation strategy, and evaluation methodology for building a system that transforms the "Text-to-SQL" academic problem into a viable business utility.

2 Objectives

2.1 Real-World Problem

In modern data-driven organizations, the volume of data is exploding, but the ability to extract insights remains concentrated in the hands of a few technical experts (Data Analysts and Data Scientists). Business stakeholders—marketing managers, operations leads, and executives—often have critical questions but lack the SQL skills to answer them directly. This creates a significant bottleneck: simple questions ("What were total sales last Q4?") can take days to answer as they sit in ticket queues, delaying decision-making and reducing agility. Furthermore, when stakeholders attempt to use generic LLMs (like ChatGPT) for this task, they cannot easily provide the model with the secure access to the full database schema required for accurate answers.

2.2 Target Users

- **Primary Users: Business Stakeholders.** These are domain experts (e.g., a Regional Sales Manager) who understand the *business logic* but not the *relational logic*. They need immediate answers to ad-hoc questions to inform strategy.
- **Secondary Users: Junior Data Analysts.** Analysts can use the Copilot to "jump-start" their work, having the system draft complex joins or boilerplate SQL which they then refine, significantly increasing their productivity.

2.3 Innovation Layer

Our project goes beyond standard "Text-to-SQL" tutorials by implementing a multi-agentic architecture inspired by recent NeurIPS 2025 research. The core innovations include:

1. **Semantic Metadata Layer (MAIA Adaptation):** We do not just pass raw table definitions (DDL) to the LLM, which are often cryptic (e.g., `t_ord_dt`). Instead, we implement a "Semantic View" layer in Snowflake that maps technical columns to verbose business concepts (e.g., "Transaction Date"), significantly reducing semantic ambiguity.

2. **RAG-based Schema Linking (X-SQL Adaptation):** A common failure mode for Text-to-SQL is ”context window overload,” where feeding an LLM 100 table definitions causes it to get confused. We use a dedicated ”Schema Linker” agent powered by Vector Search to dynamically retrieve only the top 3-5 relevant tables for a specific question.
3. **Self-Correction Loop:** We implement a ”Validation Agent” that runs a ‘EXPLAIN‘ plan on the generated SQL within Snowflake. If the database returns a syntax error (e.g., ”Column not found”), the agent captures the error, feeds it back to the LLM, and requests a correction—transparently to the user.

3 Related Work (NeurIPS 2025)

Our system design is directly informed by three cutting-edge papers from NeurIPS 2025 that address key challenges in Text-to-SQL.

3.1 Chatting With Your Data (MAIA)

Paper: *Chatting With Your Data: LLM-Enabled Data Transformation for Enterprise Text to SQL* [Liu and et al., 2025].
Summary: This paper introduces MAIA, a framework that solves the ”schema gap” by transforming rigid, normalized database schemas into a semantic layer that aligns with how humans think and speak. The authors argue that the primary cause of Text-to-SQL failure is not the LLM’s reasoning ability, but the mismatch between ”User Speak” and ”Database Speak.”
Project Integration: We adopt this philosophy by building a dedicated ‘METADATA‘ schema in Snowflake. For every table in our Olist dataset, we will generate (using a helper LLM) a JSON-based description that includes:

- A natural language description of the table’s purpose.
- Synonyms for column names (e.g., `zip_code_prefix` → ”Postal Code”, ”Zip”).
- Categorical value examples (e.g., ”Payment types can be: credit_card, boleto, voucher”).

This metadata serves as the retrieval context for our RAG system, rather than the raw DDL.

3.2 OmniSQL: Synthetic Data at Scale

Paper: *OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale* [Tang and et al., 2025].
Summary: OmniSQL demonstrates that high-quality synthetic training data can outperform human-annotated data. They generated 2.5 million text-SQL pairs to robustly train models, showing that diversity in SQL complexity (nested queries, window functions) is key to generalization.
Project Integration: We will use a similar synthetic generation pipeline to create our **Evaluation Dataset**. We will write a script that iterates through our schema and prompts an LLM to ”imagine 5 hard business questions a CEO would ask about this data.” We will then verify the SQL for these questions manually. This results in a ”Golden Dataset” of 50-100 questions that we use to quantitatively measure our system’s accuracy, ensuring we aren’t just relying on anecdotal ”it looks good” testing.

3.3 X-SQL: Expert Schema Linking

Paper: *X-SQL: Expert Schema Linking and Understanding of Text-to-SQL with Multi-LLMs* [Wang and et al., 2025].
Summary: X-SQL decomposes the problem into specialized agents: one for finding the right

tables (”Schema Linking”) and one for writing the query. This ”divide and conquer” approach reduces hallucinations by preventing the SQL-writing LLM from seeing irrelevant table noise.
Project Integration: Our architecture explicitly separates the ”retrieval” step from the ”generation” step. We will index our table metadata into a **Snowflake Cortex Vector Search** index. When a query arrives, Step 1 is ”Find relevant tables.” Only the output of this step (e.g., ‘orders‘, ‘customers‘) is passed to Step 2 ”Generate SQL.” This aligns perfectly with the X-SQL findings that focused context improves accuracy.

4 Data Sources

We have selected datasets that represent realistic enterprise complexity (Relational Schemas) and standard benchmarks.

4.1 Primary: Brazilian E-Commerce (Olist)

Source: Kaggle (`https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce`)
Description: A rich relational dataset containing 100k orders from 2016-2018. It features 9 connected tables: `orders`, `customers`, `order_items`, `payments`, `products`, `sellers`, `geolocation`, `product_category_name_translation`, and `reviews`.
Justification: This dataset is ideal because it is *highly normalized*. To answer a question like ”What is the average review score for ’Health & Beauty’ products sold in 2017?”, the system must join at least 4 tables (`reviews`, `orders`, `order_items`, `products`). This complexity creates a genuine need for an AI assistant, as writing such a query manually is tedious even for experts.
Snowflake Ingestion Strategy:

- **Stage:** Upload CSVs to Snowflake Internal Stage (‘@OLIST_STAGE‘).
- **Load:** Use ‘COPY INTO‘ commands to populate ‘RAW‘ schema tables.
- **Data Quality Checks:** Before analysis, we implement a **Validation Step** using Snowflake **Data Metric Functions**. We automatically check for:
  - Null values in critical columns (e.g., `order_id`, `price`).
  - Referential integrity violations (e.g., an `order_item` referencing a non-existent `product_id`).
  - Time range anomalies (e.g., orders dated in the future).

  This ensures our AI doesn’t learn from bad data.
- **Modeling:** We will strictly enforce Primary Keys (e.g., `order_id`) and Foreign Keys in our definition DDL. While Snowflake doesn’t enforce these constraints at write time, defining them provides critical hints to the Cortex LLM about how tables relate.

4.2 Baseline: Superstore Sales

Source: Kaggle (`https://www.kaggle.com/datasets/vivek468/superstore-dataset-final`)
Description: A single-table dataset (denormalized) representing retail sales.
Usage: Used for initial ”Hello World” testing of the pipeline to verify latency and UI responsiveness without join complexity.

4.3 Reference: Spider (Yale)

Source: Hugging Face (`https://huggingface.co/datasets/xlangai/spider`)
Description: A massive cross-domain text-to-SQL dataset.
Usage: We will index complex SQL patterns from Spider (e.g., nested subqueries, 'HAVING', 'CASE WHEN') into our Vector Store. These serve as "Few-Shot Examples" retrieved at runtime to teach the LLM how to handle complex logic.

5 Methods, Technologies & Tools

5.1 System Architecture

The system is built entirely around the **Snowflake Data Cloud** to ensure security, scalability, and unified governance.

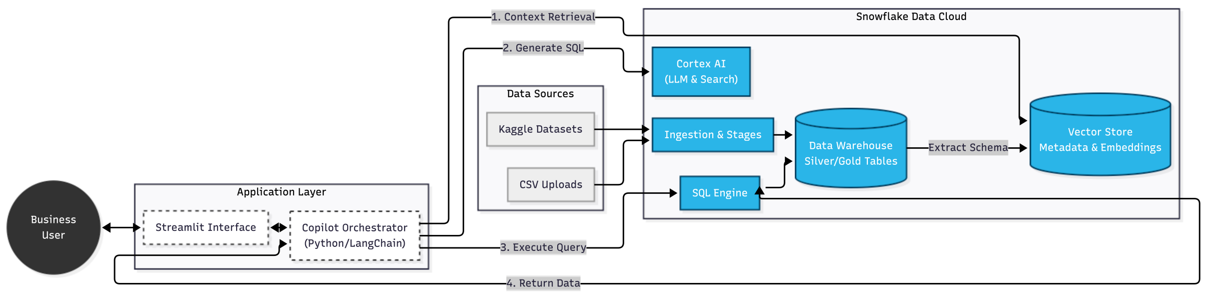

Figure 1: System Architecture: From User Query to SQL Execution

The pipeline follows a sequential "Agentic" flow:

1. **User Input:** The user asks a question via the Streamlit UI.
2. **Context Retrieval (RAG):** The system embeds the user's question and searches the Vector Store for (a) relevant tables and (b) similar "Golden Queries" from our training set.
3. **Prompt Assembly:** A dynamic prompt is constructed including the user question, the filtered schema definitions, and the few-shot examples.
4. **SQL Generation (Cortex):** The prompt is sent to Snowflake Cortex (running Llama 3 70B or Mistral Large).
5. **Validation (Self-Correction):** The generated SQL is dry-run using `EXPLAIN`. If valid, it executes. If invalid, the error is fed back to the LLM for a retry (max 3 retries).
6. **Visualization:** The result set is returned to Streamlit, which automatically selects the best chart type (Bar, Line, Scatter) based on the data types (e.g., TimeSeries vs Categorical).

5.1.1 Detailed Agent Design

To achieve high accuracy, we move beyond a simple "Chain" (A $\rightarrow$ B $\rightarrow$ C) to an "Agentic" workflow where steps can loop or branch.

- **Schema Linker Agent:** This agent receives the user query and performs a *hybrid search* (Keyword + Semantic) against the Vector Store. It returns a list of candidate tables. Crucially, it filters out tables with high semantic overlap but low relevance (e.g., 'geolocation' is often retrieved but rarely needed for sales totals).
- **SQL Generator Agent:** This agent acts as the "Coder." It is prompted with a specific "Role" (e.g., "You are a Senior Snowflake Data Engineer") and provided with the *Semantic Views* of the linked tables. It is explicitly instructed to use Snowflake-specific syntax (e.g., `TO_VARCHAR`, `DATE_TRUNC`).
- **Validation Agent:** This agent acts as the "Reviewer." It takes the generated SQL and runs a `EXPLAIN` query. If the `EXPLAIN` fails, it parses the error message and instructs the SQL Generator to fix it (e.g., "Error: Column 'profit' does not exist. Did you mean to calculate it as 'price - cost'?").

5.2 Data Ingestion & Storage

We use a standard **ELT (Extract, Load, Transform)** pattern:

1. **Ingest:** Data is uploaded to Snowflake Internal Stages.
2. **Load:** 'COPY INTO' moves data to 'RAW' schema tables.
3. **Transform:** Snowflake **Tasks** and **Streams** (optional) clean data types and create a 'SILVER' layer.
4. **Metadata:** A specialized 'METADATA' table stores column descriptions and "Golden Queries" for RAG.

5.3 Analytics, ML & GenAI

- **Vector Search:** We use **Snowflake Cortex Search** (or a vector-enabled table with `VECTOR` data type) to store embeddings of table schemas. This allows semantic search (e.g., searching for "revenue" matches the `payment_value` column description).
- **LLM:** We utilize **Snowflake Cortex** (accessing models like Llama 3 or Mistral) for secure, serverless inference inside the data boundary.
- **Orchestration:** A Python-based controller (using LangChain or simple function calls) manages the flow. We specifically use **Snowflake Notebooks** as the development environment for these orchestration scripts, allowing us to version control the logic directly in git while executing close to the data.
- **Cortex Functions:** We leverage specific Snowflake Cortex functions such as `COMPLETE` (for text generation) and `EMBED_TEXT_768` (for creating vector embeddings of our schema descriptions). By using these built-in primitives, we avoid the complexity of managing external API keys and data egress fees.

5.4 User Interface Design

The frontend is built using **Streamlit**, chosen for its native integration with Snowflake and rapid prototyping capabilities. The UI features:

- **Chat Interface:** A familiar "ChatGPT-style" message history where users can ask follow-up questions (e.g., "Now filter that by Q3").

- **Transparency Toggle:** An "Explain Logic" button that, when clicked, reveals the intermediate steps: the retrieved tables, the raw generated SQL, and the execution plan. This builds trust with technical users.
- **Visualization Library:** We use 'altair' for declarative statistical visualization. The system heuristically maps query results to chart types:
  - Time series data (Date + Metric) → Line Chart.
  - Categorical comparisons (Category + Metric) → Bar Chart.
  - Geographic data (Lat/Lon + Metric) → Map (using 'pydeck').

5.5 Infrastructure & Deployment

The entire application stack is designed to be deployed within the Snowflake environment:

- **Frontend: Streamlit in Snowflake (SiS)**. This allows us to host the UI directly inside the data warehouse, meaning the data never travels over the public internet, a crucial feature for enterprise security.
- **Compute:** We utilize standard Snowflake Virtual Warehouses (XS size for development, S/M for demo) to handle the SQL execution load.

6 Implementation Plan

We have structured the project into four one-week sprints:

- **Week 1: Data & Infrastructure.**
  - Setup Snowflake account and GitHub repo.
  - Ingest Olist and Superstore datasets.
  - Define "Semantic Views" and write table descriptions.
- **Week 2: RAG Pipeline Development.**
  - Implement Vector Store for schema metadata.
  - Develop the "Schema Linker" agent.
  - Build the basic Text-to-SQL function using Cortex.
- **Week 3: Frontend & Refinement.**
  - Build Streamlit interface.
  - Implement the "Self-Correction" loop with 'EXPLAIN' logic.
  - Integrate visualization logic.
- **Week 4: Evaluation & Final Report.**
  - Generate "Golden Dataset" using OmniSQL method.
  - Run full evaluation benchmarks.
  - Finalize PDF proposal and documentation.

7 Risk Analysis

- **Risk: Hallucination.** The LLM might invent columns that don't exist.
- **Mitigation:** Our "Schema Linking" step reduces the search space, and the "Self-Correction" agent catches syntax errors before the user sees them.
- **Risk: Latency.** Multi-step RAG chains (Retrieve $\rightarrow$ Generate $\rightarrow$ Validate) can be slow (10s+).
- **Mitigation:** We will optimize by using smaller, faster models (e.g., Mistral 7B) for the simple Schema Linking step, reserving the large model (Llama 3 70B) only for the complex SQL generation.

8 Expected Outcomes

8.1 Live Demo

Our final presentation will demonstrate a live session where we:

1. Upload a new slice of data (e.g., "2025 Sales").
2. Ask complex questions: "Show me the top 5 product categories by revenue in São Paulo."
3. Watch the system *think* (show retrieved tables), *generate* SQL, and *render* a bar chart.
4. Demonstrate "Self-Correction" by asking a vague question and seeing the system ask for clarification or fix a syntax error.

8.2 Evaluation Metrics

We will evaluate the system quantitatively:

- **Execution Accuracy (EX):** Percentage of generated SQL queries that run without error and return the correct result set (compared to Golden SQL).
- **Logical Accuracy:** We will manually audit a sample of 20 queries to ensure the join logic (e.g., `INNER` vs `LEFT` join) is semantically correct for the business question.
- **Latency:** End-to-end time from "Submit" to "Chart Rendered."
- **Token Efficiency:** Cost per query based on prompt size (reduced by our Schema Linking strategy).

8.3 Evaluation Methodology: The Golden Dataset

To rigorously measure "Execution Accuracy," we cannot rely on ad-hoc testing. We will construct a **Golden Dataset** of 50-100 Question-SQL pairs derived from the Olist schema. This process involves:

1. **Synthetic Generation:** Using the *OmniSQL* methodology, we prompt an LLM to generate diverse questions ranging from easy ("Count orders") to hard ("Average delivery time by state").
2. **Human Verification:** We manually review the generated SQL for correctness, correcting any logical errors (e.g., ensuring `freight_value` is summed correctly).

3. **Automated Benchmarking:** We write a Python script that runs all 100 questions through our Copilot, captures the generated SQL, executes it, and compares the result set (row counts, values) against the Golden SQL results. This provides a hard metric (e.g., "82% Accuracy") that we can track as we improve the prompts.

9 Final Deliverables

1. **GitHub Repository:** A public repo containing all source code ('/src'), documentation ('/docs'), and setup scripts ('/reproducibility'): `https://github.com/ben-blake/analytics-copilot`
2. **Prototype App:** A functional Streamlit web application.
3. **Snowpark Notebooks:** Python notebooks demonstrating the data ingestion and vector indexing steps.
4. **Final Report:** A comprehensive PDF detailing our design decisions, challenges, and evaluation results.

References

Y. Liu and et al. Chatting with your data: Llm-enabled data transformation for enterprise text to sql. *NeurIPS*, 2025. URL `https://neurips.cc/virtual/2025/132553`.

Y. Tang and et al. Omnisql: Synthesizing high-quality text-to-sql data at scale. *NeurIPS*, 2025. URL `https://arxiv.org/abs/2408.12658`.

X. Wang and et al. X-sql: Expert schema linking and understanding of text-to-sql with multi-llms. *NeurIPS*, 2025. URL `https://arxiv.org/abs/2509.05899`.