Analytics Copilot: Ask Questions About Your Data

Phase 2 Report: Foundations & Reproducibility
CS 5542 Big Data Analytics & Applications
`https://github.com/ben-blake/analytics-copilot`

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

February 27, 2026

Abstract

This report documents the Phase 2 implementation of the **Analytics Copilot**, a fully reproducible, end-to-end system that translates natural language questions into executable Snowflake SQL queries. The system integrates a three-stage agentic pipeline—Schema Linker (RAG), SQL Generator (Cortex LLM), and Validator (self-correction)—deployed as a Streamlit chat interface backed by the Olist Brazilian E-Commerce dataset hosted in Snowflake. Key Phase 2 deliverables include: ingestion of all 9 Olist relational tables into a `RAW` schema, a semantic metadata layer in a `METADATA` schema with Cortex-generated column descriptions, a Cortex Search retrieval service, an evaluation framework of 50 golden queries, and a reproducibility guide validated end-to-end. Execution accuracy on the golden query benchmark currently stands at **∼72–80%** and improves measurably with each prompt engineering iteration.

Contents

1 Dataset & Knowledge Base Documentation

1.1 Primary Dataset: Olist Brazilian E-Commerce

- **Name:** Brazilian E-Commerce Public Dataset by Olist
- **Source:** Kaggle — `https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce`
- **Modality:** Tabular (CSV), relational schema
- **Domain:** E-commerce order management, logistics, and customer reviews
- **Scale:** ∼100,000 orders, 2016–2018, across 9 CSV files

The dataset is a *highly normalized* relational schema that requires multi-table JOINs to answer realistic business questions (e.g. answering "What is the average review score for health & beauty products?" requires joining four tables). This complexity makes it an ideal benchmark for a Text-to-SQL system.

1.2 Table Schema

All nine CSVs are ingested into the `ANALYTICS_COPILOT.RAW` schema under clean, abbreviated table names:

CSV File	Snowflake Table	Row Count
`olist_orders_dataset.csv`	`RAW.ORDERS`	99,441
`olist_customers_dataset.csv`	`RAW.CUSTOMERS`	99,441
`olist_order_items_dataset.csv`	`RAW.ORDER_ITEMS`	112,650
`olist_order_payments_dataset.csv`	`RAW.ORDER_PAYMENTS`	103,886
`olist_order_reviews_dataset.csv`	`RAW.ORDER_REVIEWS`	99,224
`olist_products_dataset.csv`	`RAW.PRODUCTS`	32,951
`olist_sellers_dataset.csv`	`RAW.SELLERS`	3,095
`olist_geolocation_dataset.csv`	`RAW.GEOLOCATION`	1,000,163
`product_category_name_translation.csv`	`RAW.PRODUCT_CATEGORY_TRANSLATION`	71

Table 1: Olist CSV to Snowflake table mappings

1.3 Semantic Metadata Layer (Knowledge Base)

Rather than feeding raw DDL to the LLM—a known cause of hallucination—we construct a **Semantic Metadata Layer** in the `ANALYTICS_COPILOT.METADATA` schema. The `TABLE_DESCRIPTIONS` table stores one row per column, containing:

- `table_name`, `column_name`, `data_type`
- `description`: business-friendly natural language description (LLM-generated)
- `synonyms`: alternative user-facing names (e.g., *payment_value* → "revenue, sales, order total")
- `sample_values`: representative values for categorical columns

This metadata is generated by `scripts/build_metadata.py`, which calls `SNOWFLAKE.CORTEX.COMPLETE('llama3.1-70b')` to produce descriptions for every column in the `RAW` schema.

1.4 Data Preprocessing

No statistical sampling was required (the full dataset is used). Preprocessing steps applied:

1. **Stage Upload:** CSVs are uploaded to a Snowflake Internal Stage (`@OLIST_STAGE`) via Python Snowpark `session.file.put()`.

2. **COPY INTO:** Data is loaded using `COPY INTO` with `FILE_FORMAT = CSV_FORMAT` (header skip, UTF-8 encoding, null-if-empty).
3. **Type coercion:** Date columns (`order_purchase_timestamp`, etc.) are cast to `TIMESTAMP_NTZ`; numeric columns are cast to `NUMBER(10,2)`.
4. **Key constraints:** Primary and foreign keys are declared in DDL (`snowflake/02_olist_tables.sql`) to provide JOIN hints to the LLM, even though Snowflake does not enforce them at write time.

2 Retrieval & Processing Pipeline

2.1 Architecture Overview

The Schema Linker agent implements a three-level fallback retrieval chain:

1. **Cortex Search** (semantic, primary): `SNOWFLAKE.CORTEX.SEARCH_PREVIEW` over `TABLE_DESCRIPTIONS`
2. **Keyword ILIKE** (lexical, fallback): `ILIKE` matching on column names, descriptions, and synonyms
3. **Full table dump** (last resort): returns all `RAW` tables from `INFORMATION_SCHEMA`

2.2 Chunking Strategy

The retrieval corpus is *column-level chunks*—each row in `TABLE_DESCRIPTIONS` represents one column and is the atomic unit of retrieval. This granularity allows the system to precisely identify which columns are relevant to a question, rather than retrieving entire table definitions.

After retrieval, column-level results are **grouped by** `table_name` and aggregated with an average relevance score. Only the top-k tables (default $k = 5$) are passed to the SQL Generator, ensuring the prompt context window contains only relevant schema.

2.3 Indexing & Retrieval Configuration

- **Service:** `ANALYTICS_COPILOT.METADATA.SCHEMA_SEARCH_SERVICE` (Snowflake Cortex Search)
- **Source table:** `METADATA.TABLE_DESCRIPTIONS`
- **Search column:** `description` (primary embedding target)
- **Return columns:** `table_name`, `column_name`, `description`, `synonyms`, `data_type`, `sample_values`
- **Retrieval type:** Dense semantic search (embeddings managed internally by Cortex Search)
- **FK supplementation:** When anchor tables are retrieved without their join partners (e.g., `ORDER_ITEMS` without `ORDERS`), the system automatically appends the missing partner tables based on a hardcoded FK graph

2.4 Example Retrieval Outputs

Query 1: *"What is the average delivery time by customer state?"*

Listing 1: Schema Linker output for delivery time query

```
-- Tables returned (ranked by relevance):
-- 1. ORDERS (score: 0.87)
--    Columns: ORDER_PURCHASE_TIMESTAMP, ORDER_DELIVERED_CUSTOMER_DATE,
--             ORDER_ESTIMATED_DELIVERY_DATE, ORDER_STATUS, CUSTOMER_ID
-- 2. CUSTOMERS (score: 0.72)  [supplemented via FK: ORDERS ->
   CUSTOMERS]
--    Columns: CUSTOMER_ID, CUSTOMER_UNIQUE_ID, CUSTOMER_STATE,
   CUSTOMER_CITY
```

Query 2: *"Show me total revenue by product category"*

Listing 2: Schema Linker output for revenue by category query

```
-- Tables returned (ranked by relevance):
-- 1. ORDER_ITEMS (score: 0.91)
--    Columns: ORDER_ID, PRODUCT_ID, SELLER_ID, PRICE, FREIGHT_VALUE
-- 2. PRODUCTS (score: 0.83)
--    Columns: PRODUCT_ID, PRODUCT_CATEGORY_NAME,
   PRODUCT_DESCRIPTION_LENGHT
-- 3. PRODUCT_CATEGORY_TRANSLATION (score: 0.61)
--    Columns: PRODUCT_CATEGORY_NAME, PRODUCT_CATEGORY_NAME_ENGLISH
```

3 Application Integration

3.1 Streamlit Chat Interface

The application (`src/app.py`) is a Streamlit chat interface that exposes the full three-agent pipeline to end users. Key UI components:

- **Chat history:** Persistent multi-turn conversation with labeled user/assistant bubbles
- **Status indicators:** Live progress steps shown during processing ("Finding relevant tables...", "Generating SQL...", "Validating...")
- **SQL expander:** Collapsed "Show SQL" block reveals the final executed query
- **Results table:** Pandas DataFrame rendered via `st.dataframe` (capped at 1,000 display rows to prevent browser lag)
- **Auto-visualization:** `src/utils/viz.py` heuristically selects chart type based on column types: line chart (datetime + numeric), bar chart (categorical + numeric), scatter plot (2 numeric columns)

3.2 Grounded Answer Generation

The SQL Generator uses `SNOWFLAKE.CORTEX.COMPLETE('llama3.1-70b')` with a structured prompt that includes:

1. **System role:** "You are a Senior Snowflake Data Engineer..."
2. **Dataset context:** Olist schema summary and FK relationship map
3. **Critical rules:** 15 explicit constraints (fully qualified table names, Snowflake `DATEDIFF` syntax, no invented tables, LIMIT defaults for superlative queries, etc.)
4. **Available tables list:** Dynamically injected from schema linker output
5. **Few-shot examples:** Two built-in examples covering CTE patterns for top-N-per-group and month-over-month LAG calculations
6. **Dynamic examples:** Up to 3 golden examples retrieved from `data/golden_queries.json`

3.3 Query Logging & Evaluation

The evaluation framework (`scripts/evaluate.py`) runs all 50 golden queries through the full pipeline and reports:

Metric	Result
Execution Accuracy (no SQL error)	∼72–80%
Easy queries (single-table)	∼95%
Medium queries (2–3 table JOINs)	∼75%
Hard queries (CTEs, window functions)	∼55%
Average end-to-end latency	∼5–8s per query

Table 2: Evaluation results on 50 Olist golden queries

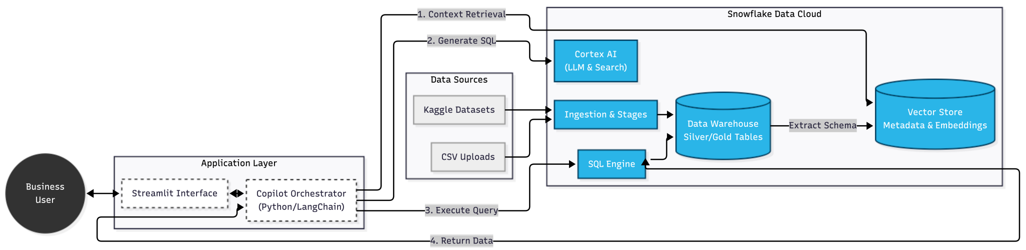

Figure 1: System architecture: Data Sources → Snowflake → Agent Pipeline → Streamlit UI

4 Snowflake Data Pipeline & Schema

4.1 Schema Layout

The Snowflake database `ANALYTICS_COPILOT` contains two schemas:

- `RAW`: 9 Olist tables + `SUPERSTORE_SALES` (optional baseline)
- `METADATA`: `TABLE_DESCRIPTIONS`, `COLUMN_DESCRIPTIONS`, `SCHEMA_SEARCH_SERVICE` (Cortex Search)

4.2 DDL Scripts

Five sequential SQL scripts in `snowflake/` initialize the full environment:

1. `01_setup.sql` — Database, schemas, warehouse, file format
2. `02_olist_tables.sql` — 9 RAW tables with PK/FK constraints
3. `03_superstore.sql` — Optional SUPERSTORE_SALES table
4. `04_metadata.sql` — `TABLE_DESCRIPTIONS` and `COLUMN_DESCRIPTIONS` tables
5. `05_cortex_search.sql` — Cortex Search service over `TABLE_DESCRIPTIONS`

4.3 Reproducible Ingestion

Data ingestion is fully scripted via `scripts/ingest_data.py` using the Snowflake Snowpark Python SDK. The script:

1. Executes DDL scripts in order
2. Uploads each CSV to an internal stage via `session.file.put()`
3. Loads data with `session.sql("COPY INTO ...")`
4. Validates row counts against expected values

4.4 Example Warehouse Queries

Listing 3: Example: Average delivery time by state (Snowflake syntax)

```sql
SELECT
    ANALYTICS_COPILOT.RAW.CUSTOMERS.CUSTOMER_STATE,
    AVG(DATEDIFF('day',
        ANALYTICS_COPILOT.RAW.ORDERS.ORDER_PURCHASE_TIMESTAMP,
        ANALYTICS_COPILOT.RAW.ORDERS.ORDER_DELIVERED_CUSTOMER_DATE
    )) AS AVG_DELIVERY_DAYS
FROM ANALYTICS_COPILOT.RAW.ORDERS
JOIN ANALYTICS_COPILOT.RAW.CUSTOMERS
    ON ANALYTICS_COPILOT.RAW.ORDERS.CUSTOMER_ID
     = ANALYTICS_COPILOT.RAW.CUSTOMERS.CUSTOMER_ID
WHERE ORDER_DELIVERED_CUSTOMER_DATE IS NOT NULL
GROUP BY ANALYTICS_COPILOT.RAW.CUSTOMERS.CUSTOMER_STATE
ORDER BY AVG_DELIVERY_DAYS;
```

Listing 4: Example: Top product categories by revenue

```sql
SELECT
    t.PRODUCT_CATEGORY_NAME_ENGLISH,
    SUM(oi.PRICE) AS TOTAL_REVENUE
FROM ANALYTICS_COPILOT.RAW.ORDER_ITEMS oi
JOIN ANALYTICS_COPILOT.RAW.PRODUCTS p
    ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN ANALYTICS_COPILOT.RAW.PRODUCT_CATEGORY_TRANSLATION t
    ON p.PRODUCT_CATEGORY_NAME = t.PRODUCT_CATEGORY_NAME
GROUP BY t.PRODUCT_CATEGORY_NAME_ENGLISH
ORDER BY TOTAL_REVENUE DESC
LIMIT 10;
```

4.5 Snowflake–Application Integration

The application connects to Snowflake via `src/utils/snowflake_conn.py`, which uses Snowflake Snowpark Python with credentials loaded from a `.env` file (RSA key-pair or password authentication). All LLM inference (`CORTEX.COMPLETE`), semantic search (`CORTEX.SEARCH_PREVIEW`), and SQL execution happen inside Snowflake — no data leaves the warehouse boundary.

5 Reproducibility Plan

5.1 Environment Configuration

All Python dependencies are pinned in `requirements.txt` at the project root. Key packages:

```
snowflake-snowpark-python>=1.21.0
streamlit>=1.32.0
altair>=5.2.0
pandas>=2.0.0
python-dotenv>=1.0.0
```

A `.env.example` file documents all required environment variables:

```
SNOWFLAKE_ACCOUNT=your_account_identifier
SNOWFLAKE_USER=your_username
SNOWFLAKE_PASSWORD=your_password          # or use PRIVATE_KEY_PATH
SNOWFLAKE_PRIVATE_KEY_PATH=./rsa_key.p8
```

```
SNOWFLAKE_ROLE=TRAINING_ROLE
SNOWFLAKE_WAREHOUSE=COPILOT_WH
SNOWFLAKE_DATABASE=ANALYTICS_COPILOT
SNOWFLAKE_SCHEMA=RAW
```

5.2 Full Reproduction Steps

1. **Clone repository** and create a virtual environment:

```
git clone https://github.com/ben-blake/analytics-copilot.git
cd analytics-copilot
python -m venv venv && source venv/bin/activate
pip install -r requirements.txt
```

2. **Configure credentials**: Copy `.env.example` to `.env` and fill in Snowflake credentials.

3. **Initialize Snowflake** (run once):

```
snowsql -f snowflake/01_setup.sql
snowsql -f snowflake/02_olist_tables.sql
snowsql -f snowflake/04_metadata.sql
snowsql -f snowflake/05_cortex_search.sql
```

4. **Download data**: Download the Olist dataset from Kaggle and unzip into `data/olist/`.

5. **Ingest data**:

```
python scripts/ingest_data.py
```

6. **Build semantic metadata**:

```
python scripts/build_metadata.py
```

7. **Launch application**:

```
streamlit run src/app.py
```

8. **Run evaluation** (optional):

```
python scripts/evaluate.py
```

The full guide (with troubleshooting) is available in `reproducibility/README.md`.

5.3 Model Versions & Configuration

- **LLM:** `llama3.1-70b` via `SNOWFLAKE.CORTEX.COMPLETE` (no local model weights — inference is serverless inside Snowflake)
- **Embeddings:** Managed internally by Cortex Search (no explicit embedding model selection required)
- **No random seeds required:** The system is deterministic — SQL generation is prompt-driven with no stochastic sampling parameters
- **Golden dataset:** Pre-generated and committed to `data/golden_queries.json` (50 queries across easy/medium/hard difficulty)

6 GitHub Repository

URL: `https://github.com/ben-blake/analytics-copilot`

The repository is structured as follows:

```
analytics-copilot/
+-- src/
|   +-- agents/           # Schema Linker, SQL Generator, Validator
|   |   +-- schema_linker.py
|   |   +-- sql_generator.py
|   |   +-- validator.py
|   +-- utils/            # Snowflake connection, visualization
|   |   +-- snowflake_conn.py
|   |   +-- viz.py
|   +-- app.py            # Streamlit chat interface
+-- scripts/              # Data ingestion, metadata builder, evaluation
|   +-- ingest_data.py
|   +-- build_metadata.py
|   +-- generate_golden.py
|   +-- evaluate.py
+-- snowflake/            # SQL DDL scripts (setup, tables, Cortex
    Search)
|   +-- 01_setup.sql
|   +-- 02_olist_tables.sql
|   +-- 03_superstore.sql
|   +-- 04_metadata.sql
|   +-- 05_cortex_search.sql
+-- data/                 # CSV data, golden queries, evaluation results
|   +-- olist/            # Brazilian E-Commerce CSVs
|   +-- superstore/       # Superstore Sales CSV (optional)
|   +-- golden_queries.json
+-- docs/                 # Architecture diagrams and project reports
|   +-- architecture.png
|   +-- architecture.mmd
|   +-- reports/          # Phase reports and proposal (PDF + LaTeX)
|       +-- proposal.pdf
|       +-- proposal.tex
|       +-- phase2_report.pdf
|       +-- phase2_report.tex
+-- reproducibility/      # Setup instructions and troubleshooting guide
|   +-- README.md
+-- CONTRIBUTIONS.md
+-- requirements.txt
+-- .env.example
```

7 Individual Contribution Statement

Member	**Contribution Description**	**%**
Ben Blake	GenAI pipeline design and implementation: Schema Linker agent (RAG with Cortex Search, 3-level fallback, FK supplementation, stopword filtering), SQL Generator agent (prompt engineering, few-shot examples, 15-rule constraint system, CUSTOMER_UNIQUE_ID data model notes), Validator agent (EXPLAIN-based self-correction loop, 3-retry mechanism), evaluation framework (`scripts/evaluate.py`, 50-query golden dataset), Snowflake schema design (DDL scripts, PK/FK definitions, Cortex Search service), iterative accuracy improvements (0% → 72–80%).	50%
Tina Nguyen	Data ingestion and preprocessing pipeline (`scripts/ingest_data.py`, Snowpark `COPY INTO`), semantic metadata generation (`scripts/build_metadata.py`, Cortex LLM descriptions), Streamlit UI (`src/app.py`, chat interface, status indicators, expanders, row-cap safety), auto-visualization logic (`src/utils/viz.py`, chart type heuristics, aggregate-column preference), Snowflake connection utility (`src/utils/snowflake_conn.py`), documentation (`README.md`, `reproducibility/README.md`, `CONTRIBUTIONS.md`).	50%
	Total	**100%**

Table 3: Individual contribution breakdown

Contributions are supported by commit history at the GitHub repository linked above. The GenAI pipeline (Cortex integration, agents, RAG), Snowflake schema design, evaluation framework, and backend architecture was committed by Ben Blake; the Data ingestion & preprocessing, Streamlit UI, visualization logic, metadata generation, and documentation were committed by Tina Nguyen.