PulseAI Pipeline - Phase 2

BA 882 - Team 3

GitHub: PulseAI - GitHub

Jira: PulseAI - Jira

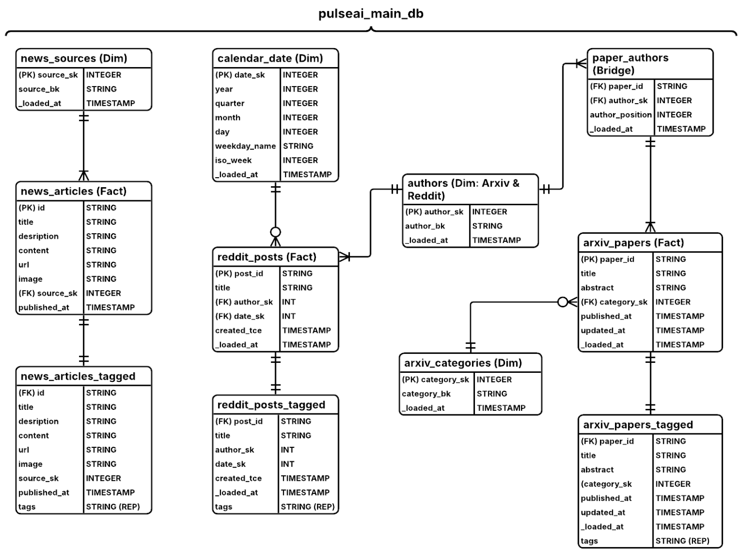
Objective: Our project aims to help users stay up to date on AI news, research, and trends by leveraging an analytics pipeline that pulls relevant data from the internet and feeds it into AI/ML workflows to produce curated summaries and dashboards. This will enable users to sift through the noise and digest curated outputs that get at the heart of the day's AI developments.

Updated Data Model: At the end of Phase 1, our main data was living in BQ tables within individual databases titled "raw_arxiv," "raw_gnews," and "raw_reddit." These tables had undergone virtually no transformations beyond applying a schema, and combined facts and dimensions with little way to connect or combine the tables.

At the end of Phase 2, we have now further transformed the data so it is more usable in analysis, and aggregated this data from all sources into a single database, "pulseai_main_db," which allows for easier access, improved visibility, and better analysis options, as the data is now cleaner and better-structured.

The main additions to the database, aside from splitting fact and dimension tables, were:

- The "paper _authors (Bridge)" table allowed us to handle the repeated-string type that arises when Arxiv papers often have multiple authors.
- The "..._tagged" tables that were a result of our ML work and are a carbon copy of our fact tables, with the sole addition of the "tags" column.



Updated Pipeline: For Phase 2, our pipeline from Phase 1 remained the same in structure, but with some minor updates to the Gnews component:

- It was not loading into BQ as we wanted, so that logic was coded into the original load function.
- We noticed there were no unique “IDs,” so we generated them using a URL-based hash and coded that in as well.

Our new, complete pipeline follows this over-simplified flow:

- Source → *Extract* → GCS → *Schema Setup* → *Load (BQ, Raw Tables)* → *Transform* → *Land (BQ, “pulseai_main_db” tables)* → *Machine Learning* → *Land (BQ, “pulseai_main_db” “tagged” tables)* → Output (Streamlit).

Like Phase 1, we leaned on *Cloud Functions* orchestrated using Airflow/Astronomer. There are three transformation functions for each source, which are combined into a single DAG that creates *“*pulseai_main_db.” The *Hugging Face* model code also includes a single function that is run on each source, using individual DAGs that are highly detailed and unique to the source's schema/characteristics.

This increased reliance on detailed DAGs is a divergence from our previous approach of coding detailed functions and using the DAGs almost exclusively as execution engines. Still, we felt this was a suitable approach at this phase, since the pipeline already has high visibility and transformation redundancy, meaning any issues that would arise during ML DAG runs would likely appear in earlier functions. Ultimately, we went this route because it leverages existing resilience without requiring us to code 9 additional ML functions.

Our Streamlit app is coded in Python and connected directly to our BQ, serving as the data source; therefore, no additional functions/pipelining were deemed necessary. This implies a “local run,” and orchestrating this process will be a priority during the final phase.

Machine Learning: In this phase, we created multiple Cloud Functions to automate extraction, loading, and transformation for Arxiv, Reddit, and GNews. These functions streamline the pipeline and prepare the data for topic modeling. We started the topic modelling by concatenating the datasets from Arxiv, Reddit, and GNews. This gave us a unified result that can be shared with both technical and non-technical audiences.

We implemented a Cloud Function that performs zero-shot text classification using a HuggingFace MNLI model. To handle model size and optimize performance, the function uses Google Cloud Storage as a model cache: it first checks if the model exists in GCS, downloads it if available, or fetches it from HuggingFace and uploads it to GCS for future use.

The model is then loaded locally into a temporary directory and exposed through a lightweight HTTP API. This setup enables us to efficiently classify incoming text into predefined categories while maintaining a serverless environment that remains fast, scalable, and cost-effective. We classify each text content into 9 business topics: Marketing, Sales, Finance, HR, Supply Chain, Customer Experience, Tech, Strategy, and Other.

Operationally, the inference step runs after data lands in BigQuery, processes only incremental rows, and writes predictions to a dedicated table for reproducibility and audit. The system is deployed in us-central1 with 32 GB RAM, 8 8-core CPU, and up to 100 concurrent instances, keeping the environment fast, scalable, and cost-effective.

Airflow DAG: Runs daily to auto-tag new articles/papers/posts from all sources. It first ensures the BigQuery target table (e.g., pulseai_main_db.reddit_posts_tagged, partitioned by created_utc) exists. It then reads only untagged rows from the original table (e.g., pulseai_main_db.reddit_posts) via a LEFT-join anti-join (WHERE t.post_id IS NULL) to prevent duplicates.

For each article/paper/post, the title is sent to the classification Cloud Function, and the returned tag(s) are inserted into the tagged table. The DAG processes in batches and includes basic retries; on a first run with no tagged table, it safely processes all articles/papers/posts.

Visualizations (Streamlit): To test our dashboard and visualizations, we created a Streamlit dashboard and deployed it locally. Our Streamlit can be better understood by viewing the visuals in the appendix or at the provided link, but our dashboard's basic functionality is as follows:

- Select reports for Arxiv, Reddit, or Gnews from a dropdown menu.
- Select the columns and download the CSV file as needed.
- Select and view the table Schema
- We have also incorporated search functionality into the dashboard, enabling keyword searches by title or abstract (e.g., it will match the input "AI" to "AI" in the title).
- Visualizations for exploratory data analysis (EDA). Users can select data sources from the dropdown and view time-series data to identify trends and patterns.

Looking Ahead

- Solve the zero-shot classifier's underdetection of the healthcare topic.
- In the final phase, we will work on the GenAI component by using it to build the AI-curated newsletter and integrate it into our workflow.
- Airflow orchestrates the run (Cloud Run Function), and aggregated views power our Streamlit dashboard for topic trends and search.
- Explore other options, like developing a React application for Phase 3. Provide greater control over the user interface and handle Gen AI integration in a more user-friendly way.
- Further database standardization and cleanup: Naming conventions, dropping unused columns, etc..