PulseAI Pipeline - Final Deliverable

BA 882 - Team 3

GitHub: PulseAI - GitHub

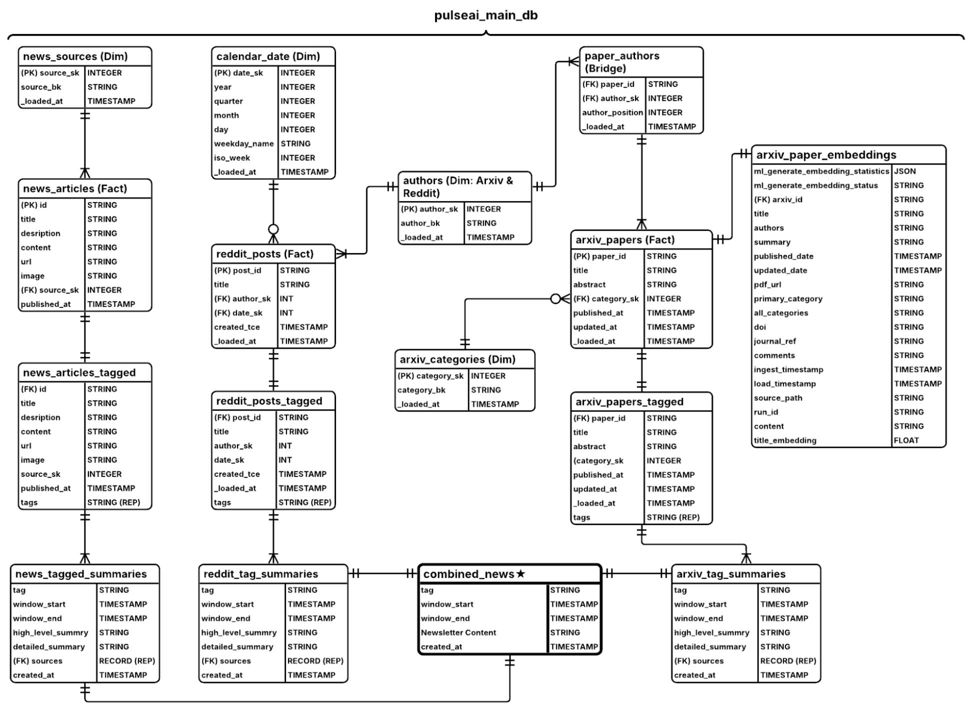
Jira: PulseAI - Jira

React App: PulseAI - App

Introduction & Business Problem: Today, the volume of new AI content across cutting-edge research, major news, and public discourse is too large for any one person to consume and understand. However, in order to stay informed and adaptable to these constant changes, understanding the current state of AI is crucial. Having finally executed the vision of creating a condensed, user-friendly newsletter that captures "The week in AI," the impact of this newsletter is clear:

The PulseAI newsletter bridges this gap and gives readers the full picture of where AI is and where it is going, without requiring significant time or omitting key developments. Thus, PulseAI readers can expect to need only a few minutes each week to stay informed and help make sound decisions in areas of their lives where AI has an impact.

Final Data Model: From Phase 2 to the final state of our project, we added only five new tables: "arxiv_paper_embeddings" (For semantic search), the three "..._summaries" tables for each source, and the "combined_news" table that feeds our React apps. All core tables live inside of "pulseai_main_db" and can be seen below:



Final Pipeline Orchestration: We use Astronomer-managed Airflow to orchestrate all Cloud Functions and ML jobs in one place, with clear dependencies between raw ingestion, transformation, topic tagging, and GenAI summarization.

Three ETL pipelines ("..._etl_pipeline") run at 7:00 PM daily to pull new content from each source and load it into raw BigQuery tables. The "pulseai_main_transformation_pipeline" runs at 8:00 PM daily, building the fact and dimension tables in "pulseai_main_db." Then, at

9:00 PM daily, three tagging pipelines (…_tagging_pipeline) run, iteratively reading over untagged rows via an anti-join pattern and writing a topic prediction back to "…_tagged tables" until all records are tagged.

The summary-creation workflows run weekly on Saturday mornings, where their "backbones" are tested and optimized prompts embedded within Cloud Functions. First, at 12:00 AM, the "newsletters_weekly_dag" pulls top-tagged articles across all sources, groups them by topic, and sends aggregated text to a GenAI Cloud Function using the Gemini 2.5 Flash model. This function produces "high-level" and "detailed" summaries, attaches topics and source lists, and stores everything in "..._summaries" tables for each source. From there, "combined_newsletter_pipeline" runs at 2:00 AM. This final workflow first creates an in-flight table using a SQL query embedded in a Cloud Function that merges on "tag" and concatenates the "detailed_summary" for a given week from each source into a single cell within a single row (i.e., the *Supply Chain* "detailed_summary" for Arxiv, Gnews, and Reddit are combined, verbatim, into a single cell). Next, each of those nine combined tag summaries are summarized by Gemini 2.0 Flash before the final "summaries of summaries" land in the "combined_news" table, which is connected directly to our React apps.

For our chosen pipeline scheduling/timing, the decision was based on a balance of business logic and technical considerations. Specifically, executing the source-level EtLT and tagging on a daily basis was primarily driven by technical considerations: We believed that extracting and transforming a week's worth of data across three sources in parallel could unnecessarily stress our pipeline and compute resources while offering no material benefit compared to doing so daily. Conversely, the summaries are created weekly because we did not think there would be enough content to produce a deeply informative daily or even bi-weekly newsletter, and because we also considered how costly it would be to run the LLM summary process 7x weekly versus 1x.

Overall, this design converts raw multi-source content into a curated, low-cost, and easy-to-consume product. Further, the staggered schedule keeps data fresh for weekly newsletters, provides adequate time buffers for processing, avoids unnecessary reprocessing, and helps control costs.

The final pipeline flow is below. The new additions are *italicized*:

- 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) → *Summarize (LLM) → Land (BQ, "..._summaries" tables) → Merge (SQL) & Summarize (LLM) → Land (BQ, "combined_news") → Output (React)*

On a final note, in the last phase of the project, we encountered roadblocks when deploying our DAGs for the LLM work due to new authentication requirements and related syntax that caused failures when deploying to Astronomer, even though the locally tested functions were confirmed to work. While we did get them to run in Astronomer, eventually, it made us question whether the Cloud Function → DAG structure we used for all our work was optimized for every workflow. While Astronomer/Airflow proved valuable for most of our project, we found that the extra work required to write and test some functions and their DAGs was unnecessary. This is why the in-flight table for the “combined_newsletter_pipeline” was created using an embedded SQL query rather than a table-creation/schema setup function + DAG.

ML Workflows: For topic classification, we used a zero-shot classifier based on a HuggingFace MNLI model to assign each article to one of nine business topics (Marketing, Sales, Finance, HR, Supply Chain, Customer Experience, Tech, Strategy, Other). The model runs as a Cloud Function with weights cached in GCS, and inference is triggered incrementally from BigQuery, writing results into the “…_tagged tables.” We have not tweaked the HuggingFace model, as it is pre-trained for text classification, and we are trying to classify a variety of new text coming in from various data sources. In the future, when we are adding more categories, it will be helpful to run this pre-trained model. Additionally, while the classification is not perfect, it significantly helps ensure the pipeline runs efficiently, as providing the model with nine topically similar aggregated summaries is a lighter workload than providing it with three source-level aggregated summaries, which are much larger and have significantly lower topic homogeneity.

Evaluation of Gen AI: Since our newsletter summarizes information extracted from the Internet, prompt engineering was vital for our project. We used an LLM as a judge to evaluate prompts: a ‘pro’ version of the LLM (Gemini 2.5 pro) evaluates the results from the ‘base’ version (Gemini 2.0 flash), which generates summaries in production. We tested different prompts with the same test samples, and asked the ‘Pro’ model to assign a hallucination score (1-10) to the outputs of the ‘base’ models. A higher hallucination score indicates higher hallucinations, and a lower score indicates higher accuracy to the source material. The prompt with the lowest score (i.e., the highest accuracy) was selected and deployed in the final production environment.

To make this more robust, we also added a completeness and conciseness score. As before, the pro version of the LLM compared and ranked summaries by this metric, rewarding the best. The average score (highest is best) for the samples revealed the best prompt.

This was augmented by human-in-the-loop evaluation, as LLMs can’t be relied on solely for production-grade processes. The prompts were approved only after a team member validated the results.

Results

- Ingested articles/posts daily from Arixv/Reddit/GNews automatically
- Tracked volumes and trends over time for each topic
- Combined three sources and generated detailed summaries along with category tags
- React App supports search, basic downloads, and exploratory visualizations

Limitations

- Solve the zero-shot classifier’s underdetection of healthcare topic
- Hallucinations can never be completely prevented due to Gen AI usage
- We could ingest more data from these current sources
- Add more sources and give the model more context around it

Appendix

Pipeline Summary Table

Pipeline Name	**Scheduled (Y/N)**	**If scheduled, when does it run?**	**Indicate the dependencies (e.g. pipeline) for this entry**	**Notes (if applicable)**
arxiv_etl_pipeline	Yes	0 0 * * *		Result of t in EtLT
gnews_etl_pipeline	Yes	0 0 * * *		Result of t in EtLT
reddit_etl_pipeline	Yes	0 0 * * *		Result of t in EtLT
pulseai_main_transformation_pipeline	Yes	0 1 * * *	arxiv_etl_pipeline; gnews_etl_pipeline; reddit_etl_pipeline	Move from raw tables to main db (Result of T in EtLT)
arxiv_paper_tagging_pipeline	Yes	0 2 * * *	pulseai_main_transformation_pipeline	Using Hugging Face model
news_article_tagging_pipeline	Yes	0 2 * * *	pulseai_main_transformation_pipeline	Using Hugging Face model
reddit_post_tagging_pipeline	Yes	0 2 * * *	pulseai_main_transformation_pipeline	Using Hugging Face model
combined_newsletter_pipeline	Yes	0 8 * * 6	newsletters_weekly_dag	Summarize at souce-level and land in individual source tables
newsletters_weekly_dag	Yes	0 0 * * 6	arxiv_paper_tagging_pipeline; news_article_tagging_pipeline; reddit_post_tagging_pipeline	Create single summary for each tag using summaries from all three sources