Vibe Radar - Technical Proposal

Marc Parcerisa, Walter Troiani, Mateja Zatezalo

March 2025

1 Executive Summary

Vibe Radar is a cutting-edge SaaS platform designed to provide large-scale impact analysis for product launches across the globe. With the increasing influence of social networks in shaping consumer behavior, enterprises and organizations need real-time, data-driven insights to gauge public reception, sentiment, and engagement. Vibe Radar empowers businesses by delivering deep analytics on how their products are being discussed and perceived across a growing list of social platforms.

Our solution offers two key products: a **standard analytics package** with daily or weekly impact reports, and a **real-time monitoring suite** for enterprises requiring immediate feedback on their product launches. Through sentiment analysis, demographic insights, view counts, and engagement metrics, businesses can make critical, informed decisions to optimize their marketing strategies. With our commitment to continuously expanding our social media tracking capabilities, Vibe Radar ensures that companies stay ahead of the curve in understanding their audience and maximizing product impact.

This document outlines the full scope of Vibe Radar, detailing both its functional capabilities and the technical infrastructure behind it. We begin by defining the project's objectives and core functionalities, followed by a breakdown of its architecture, technology stack, and security measures. Additionally, we cover scalability, deployment strategies, cost estimates, and plans to ensure a comprehensive understanding of how Vibe Radar delivers powerful, real-time impact analysis for businesses worldwide.

2 Scope and Objectives

Vibe Radar is designed primarily for large enterprises and government entities that require structured, real-time impact analysis of their products, policies, or campaigns across multiple social networks. While the core focus is on large-scale operations, we also plan to offer scaled-down versions with fewer networks and less frequent updates for organizations that need a more budget-friendly solution.

Current social media tracking is complex and resource-intensive, requiring dedicated teams to manually sift through vast amounts of unstructured data. Vibe Radar eliminates this challenge by providing fully automated data ingestion, processing, and transformation into clear, actionable KPIs. Companies can track metrics such as sentiment analysis, engagement rates, virality scores, and demographic insights without the need for extensive in-house analytics teams.

To enhance flexibility, Vibe Radar will offer different plans based on the types of KPIs clients prioritize. Enterprises that focus on sentiment analysis may opt out of demographic insights, streamlining their experience while still receiving the data most relevant to their objectives. Additionally, it will provide **API integrations and data export options**, ensuring businesses can seamlessly incorporate Vibe Radar's analytics into their existing dashboards and reporting tools. By offering both standard and customizable solutions, Vibe Radar aims to be the go-to platform for enterprises seeking precise, real-time insights into their social media impact.

3 Functional Specifications

Vibe Radar is designed to provide enterprises and government entities with a seamless, data-driven approach to tracking product impact across social media platforms. This section outlines the **core features**, **user roles**, and **typical use cases** for the platform.

3.1 Core Features

Here's a list of all the features that Vibe Radar will eventually provide. Due to their complexity, only a sub-set of them will be available on the Initial Release.

3.1.1 Sentiment Analysis

- Uses AI-powered natural language processing (NLP) to classify public sentiment (positive, neutral, or negative).
- Helps businesses understand audience perception of their product launches.

3.1.2 Engagement Tracking

- Monitors key engagement metrics such as views, likes, dislikes, shares, and comments across social networks.
- Tracks the volume of conversation and general buzz surrounding a product.

3.1.3 Geographic Insights (Not planned for Initial Release)

- Provides location-based sentiment and engagement data.
- Helps businesses understand how different markets react to a product.

3.1.4 Predictive Analytics

- Identifies patterns in audience engagement to provide forecasts on potential impact.
- Helps businesses refine their marketing strategies by anticipating audience response.

3.1.5 Historical Data Analysis

- Maintains a data lake of collected social media posts for trend analysis.
- Stores aggregated KPI data indefinitely, allowing long-term access with a small data-holding fee.

3.1.6 Real-Time Monitoring (Premium Tier)

- Provides instant sentiment and engagement insights for companies launching new products.
- Uses lightweight AI models optimized for speed rather than deep analytics.
- Designed for enterprises needing rapid feedback on launch performance.

3.1.7 Customizable Tracking Requests

- Companies submit petitions specifying:
  - **Product name**
  - **Keywords to track**
  - **Tracking tier** (determining frequency, networks covered, and analysis depth)
- The Vibe Radar team then sets up and activates the tracking system, ensuring a high-end and personalized experience.

3.2 User Roles and Permissions

Vibe Radar will offer a **hierarchical role-based access system** to accommodate different users within a company:

Role	Permissions
Admin	Manages company account, creates user roles, oversees all tracking cases.
Manager *(Tentative)*	Creates/deletes tracking cases, accesses and downloads KPIs. *(May be removed to keep tracking setup personalized.)*
Analyst	Views and downloads KPI data, accesses information about tracked products.
Executive	Limited access to KPI dashboards and downloadable reports only.

At launch, **product tracking cases will be managed directly by Vibe Radar's team** to ensure a premium, hands-on experience. Over time, a **self-service model** may be introduced for companies wanting more control.

3.3 Use Cases

Vibe Radar has a vast amount of use cases for all kinds of companies and government agencies, large or small:

3.3.1 Product Launch Monitoring for Enterprises

A multinational company launching a new smartphone uses Vibe Radar's **real-time analytics** to monitor sentiment shifts, engagement spikes, and geographic trends during the first 48 hours. Based on data insights, they adjust their marketing strategy dynamically — amplifying positive messaging in certain regions while addressing concerns in others.

3.3.2 Public Sentiment Tracking for Government Agencies

A government agency introduces a new public policy and wants to measure its reception across different demographics. Using Vibe Radar's **historical analysis and sentiment tracking**, they gauge public opinion shifts over weeks or months, allowing for adjustments in messaging and strategy.

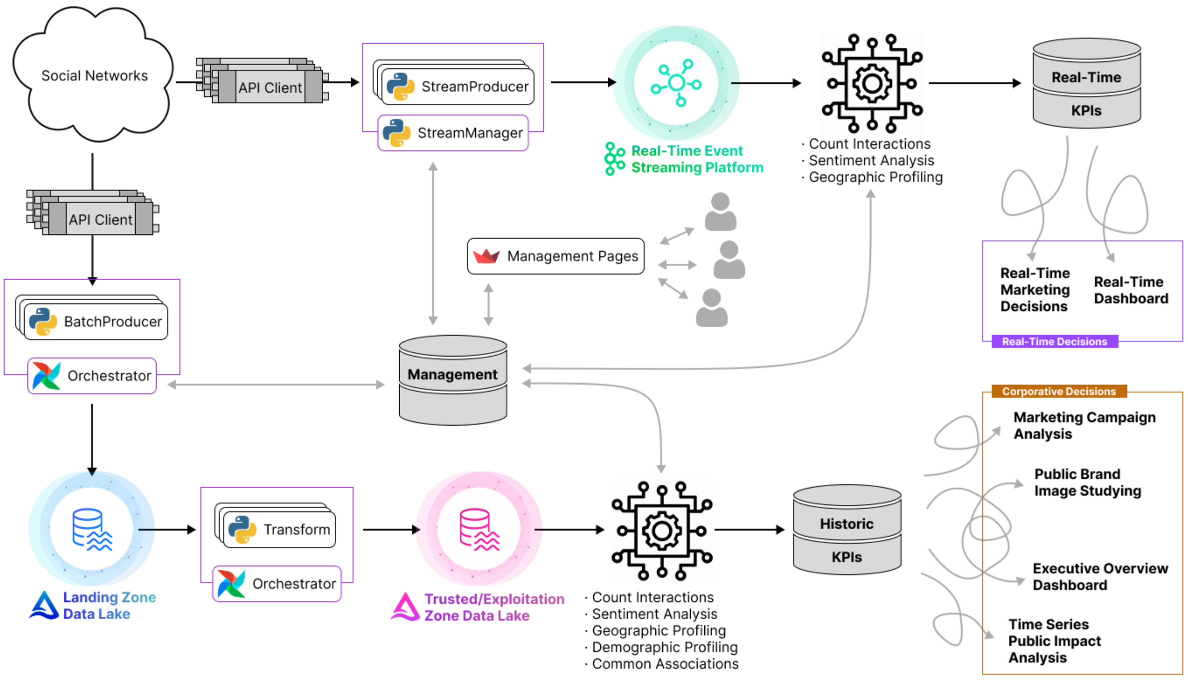
3.3.3 Competitive Benchmarking for Brands

A footwear company wants to compare audience engagement between their latest product and a competitor's. Vibe Radar tracks both brands across multiple social networks, providing a clear engagement and sentiment analysis comparison to refine future marketing campaigns.

By offering **deep social media insights, customizable tracking tiers, and a premium, high-touch experience**, Vibe Radar positions itself as the go-to platform for businesses that need precise, real-time intelligence on their product launches and brand perception.

4 Technical Architecture

In this section, we provide a high-level overview of Vibe Radar's technical architecture, highlighting the core components and how they interact to deliver real-time insights and analytics. Below, you'll find a diagram illustrating the data flow within the system, followed by an in-depth explanation of the technology stack, integration capabilities, and how data is processed and stored.



The overall data flow can be broken down into two distinct paths: the **Cold Path** and the **Hot Path**. The **Cold Path** handles the batch extraction, loading, transformation, and analysis of historical data at the frequency defined by the client. In contrast, the **Hot Path** focuses on the extraction, transformation, and real-time analysis of live data from social networks. A centralized governance database infrastructure is employed to manage, control, and govern the system, as well as store metadata associated with both data paths.

4.1 Architectural Design

This section provides a detailed breakdown of the project's architecture, key abstractions, design choices, and the technology stack used across different phases.

For the provided toy example of the full pipeline, **Docker Compose** was used to locally spin up the entire stack via a simple and common declarative *infrastructure as code* language. This approach provided an *isolated, scalable, and reproducible* environment, enabling integration of multiple software dependencies. By packaging each service in a container, a cloud-centric architecture was pursued, aiming at implementing the *Microservices* model.

4.1.1 Orchestration

The orchestration of most of this system is managed using **Apache Airflow**, chosen for its ease of use, intuitive graphical interface, and robust feature set, including task scheduling, advanced logging, and complex dependency management. Airflow ensures a structured and automated workflow, improving scalability and fault tolerance. When setting it up, it requires a connection to a *PostgreSQL* database where it store its own metadata.

Airflow's purpose is to execute tasks that end eventually in an orderly manner, hence the *directed graph* paradigm it implements. However, some of the tasks that require execution must run indefinitely - mainly, the streaming tasks. Thus, for the small-scale example realized, a simple custom manager script was developed, which provides control over the streaming producer processes via a RESTful API.

On top of those, a **Streamlit** web page was developed as a simple GUI for users to control the whole system. From there, *Batch Producing DAGs* (from *Airflow*) can be triggered, *Streaming Pods* (from the *Streaming Manager API* can be started and stopped, etc.

4.1.2 Data Ingestion

For both the Hot and Cold paths, the following three abstractions have been defined:

1. **API Clients**
   Given the immense variation on API-interfaces/scrappers of all the different social networks, a set of *Python* objects were be defined with the sole purpose of providing a high-level, interface to access both historic and real-time data from all the available data sources. They were coded in *Python*, to provide an ease of development for the further tasks, as well as unique interfaces to communicate with the social networks.
   They also provide further separation of responsibilities, easing off the process of eventually adding new social networks to the watch-list.

2. **Batch Producers**
   A *Batch Producer Python* object was developed for each of the social networks to be tracked. They are built around a single *produce* method, which, given a *query*, and a *time range*, fetches all the data available from the social network API (using the aforementioned *API Client*) and writes it into a *Staging Storage Area*, which acts as a *buffer* before being loaded into a *Landing Data Lake*.
   An auxiliary *Task* object was also developed as an abstraction that, given a list of *queries* and a *time range*, spins up all the *Batch Producers* for all the social networks and *queries*.

3. **Stream Producers**
   Similar to *Batch Producers*, a *Stream Producer Python* object was developed for each social network. This time, however, their *produce* function is meant to be an *infinitely-running daemon* script, which constantly watches the social network for new data related to the *query*, and creates *events* into their respective *Kafka* topic.
   Again, an auxiliary *Task* object was also developed to spin up all the *Stream Producer* processes given a list of *queries*.

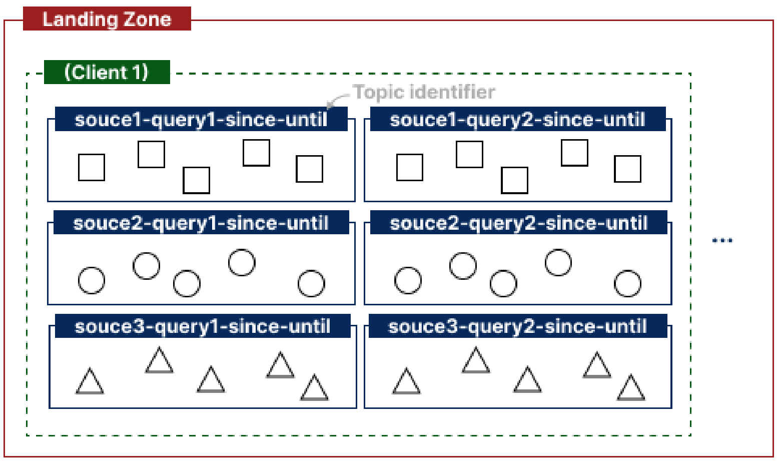
4.1.3 Metadata and Management databases

These services refer to a centralized control of cases, as well as other metadata needed for the management of the pipeline. As they require less frequent access of data, as well as holding lower volumes of it, a *PostgreSQL* instance should be more than enough. However, for the toy example provided, this data is currently being stored in example *json* files.

In these databases, we intend to store client information, user and role information, product case definitions, as well as key-words to track for each product, tiering, access to dashboards or to different APIs, etc.

4.1.4 Landing Zone Data Lake

This is probably the database that will contain the most information and the highest traffic. In order to guarantee its availability, as well as speed, it is structured as follows:


Landing Zone
(Client 1)
Topic identifier
souce1-query1-since-until
souce1-query2-since-until
souce2-query1-since-until
souce2-query2-since-until
souce3-query1-since-until
souce3-query2-since-until
...

Generally speaking, all data is stored *as-is* inside a topic (or a bucket) identified by a combination of four elements:

- **Source:** An identifier of the social network the data came from. All data coming from the same source has roughly the same structure.

- **Query:** A representation of the query that generated that data. To ensure consistency, instead of identifying the bucket with the query text, the query string is hashed via the *sha256* algorithm, converted into a *hex-based* string, and a sub-section of it is used.

- **Since:** The date and time string representation of the lower boundary for the creation time of the data inside the topic, in the following format: YYYYMMDDHHMMSSfffff

- **Until:** The date and time string representation of the upper boundary for the creation time of the data inside the topic, in the following format: YYYYMMDDHHMMSSfffff

Given the variability on the kinds of data generated from a social network, many kinds of documents need to be stored. The explained structure of storing the data was implemented in a two-step manner.

1. The *Batch Producers* write the data into a local *staging* folder, structured in the same manner as in the target data lake. In this stage, *.jsonl* files are used for the *Semi-Structured Data*, and other binary files (such as *mp4*) are used for videos, audio, images, and other unstructured data. This folder serves as a *buffer* for the data lake, which reduces the network traffic, as well as the potential overhead of creating and maintaining connections to other systems.

2. *Delta lake* is a storage layer that provides *ACID* transaction guarantees between batch and streaming data pipelines, enabling robust data management and exploration capabilities. In this stage, the semi-structured data from *.jsonl* files, such as Twitter and Bluesky posts among others, is directly stored in the delta lake. This allows us to maintain a historical record of all data changes. As the delta lake does not support unstructured data, we need a cloud storage to store videos, thumbnails, and audio. Proposed solutions include *Amazon S3* or *minIO*. The metadata of the unstructured data and the respective pointers were also stored in the delta lake. Extra storage to store unstructured data will be added during the second part of the project due to time constraints.

4.1.5 Real-Time Event Streaming

An event streaming platform provides decoupling of the analysis programs and algorithms from the sources of the data. In the toy example provided, **Apache Kafka** was used as a broker of events created by a fleet of *watcher* programs, which we call the *Stream Producers*. Whenever a post is published on a social network, the *watcher* will create a new event inside a *Kafka topic*. On the other side, a fleet of *Stream Consumers* will subscribe to said *topics*, and handle them as they appear, analyzing them in real time.

Inside Kafka, events are organized into the divisions similar to the ones in the *Landing Zone Data Lake*. However, due to the real-time nature of this path, topics will only contain the following two keys:

- **Source:** An identifier of the social network the data came from. All data coming from the same source has roughly the same structure.

- **Query:** A representation of the query that generated that data. To ensure consistency, instead of identifying the bucket with the query text, the query string is hashed via the *sha256* algorithm, converted into a *hex-based* string, and a sub-section of it is used.

Kafka transparently creates partitions of the data inside the topics, to create redundancy, as well as providing fast read speeds. For the toy example, as it doesn't use multiple nodes for the storage layer, a replication factor of 1 was left as default.

4.1.6 Trusted Zone Data Lake

To be defined

4.1.7 Historic KPIs Database

Analyzed data coming from the *Cold Path* will be stored in a *PostgreSQL* data warehouse, which will allow for easy time series analysis. If the volume of data were to grow substantially to the point of outgrowing the proposed engine, data could be migrated into *Amazon Aurora* or other high-throughput alternatives.

4.1.8 Real-Time KPIs Database

Similar to the above *Historic KPIs Database*, this database will hold the results of the *Hot Path* analyses, and will allow for the creation of real-time dashboards. This database is expected to require a higher amount of write operations per second, so it will most likely be implemented in *Amazon Aurora*.

4.1.9 Analytics

To be defined

4.1.10 Dashboarding

To be defined

4.1.11 APIs for client access

To be defined

5 Deployment and Scalability

5.1 Deployment

For users interested in deploying the toy VibeRadar example, comprehensive documentation and setup instructions are available in the official open-source repository, which is licensed under **GPLv3**, ensuring that all non-creator deployments remain **open-source and non-commercial**.

GitHub Repository: `https://github.com/eZWALT/BDM-Big-Data-Management`

5.1.1 Basic Setup

Deployment is streamlined through **Docker Compose**, requiring minimal configuration to enable a fast, easy and virtually isolated deployment:

1. Set up social media **API keys** and **Airflow Authentication** in your environment variables. See the `README.md` file for details about which environment variables are required.

2. Build/Pull and start all containers via:

```
docker-compose up -d --build
```

5.2 Scalability

The toy VibeRadar example architecture is designed for rapid deployment and scalability using a fully **Dockerized** environment spanning **10+ pre-configured containers**. This allows for easy setup on any system supporting **Docker**.

5.2.1 Current Scaling Approach

Docker Compose manages the containerized services, supporting multi-container deployments. While Docker Compose allows multi-instance scaling, it lacks true **multiple host auto-scaling** (e.g., dynamic resource allocation based on demand using multiple commodity pc's). Therefore, although some scaling capabilities exist, they are mainly vertical scaling, as opposed to the cloud-standard horizontal scaling. However, the entire project is designed around the *Microservices* paradigm, which are specifically designed to enable horizontal scaling.

5.2.2 Why Not Kubernetes?

Although **Kubernetes (K8s)** could provide robust auto-scaling and high availability, it is beyond the scope of this project. However, on a production SaaS deployment, it, or a simpler Docker-swarm, would most likely be used, which would provide redundancy, high availability, and even elastic horizontal scaling.

5.3 Cost Estimates

1. Corporate users are subject to standard VibeRadar pricing tiers.

2. Non-corporate users can deploy VibeRadar free of charge under the open-source GPLv3 license.

5.3.1 Estimated Deployment Costs

For a self-hosted deployment using a single commodity PC, the cost estimate primarily considers power consumption, hardware depreciation (amortized), and essential maintenance. We provide a grossly accurate approximation based on Spanish average prices.

Assumptions:

- Single PC consuming approximately **350W**, running 24/7.
- Hardware cost amortized over **3 years** (assuming a $700 - $1200 machine).
- Regular **maintenance** includes SSD/HDD and other hardware expenses (Amortized).

Resource	Estimated Cost (per month)
Electricity (Single PC, 350W avg, 24/7)	$40 - $70
Hardware Amortization	$20 - $35
Networking/Ethernet costs	$40
Maintenance	$10 - $20
Total Estimated Cost	**$110 - $165**

Table 1: Estimated on-premise deployment costs using a single commodity PC

In order to deploy all those containers in the cloud, at least 16 cores and 16GB of RAM are needed to manage this complex software, therefore forcing the usage of fairly expensive machines (Cloud providers usually offer small machines for horizontal scaling).

Provider	Estimated Cost (per month, April 2025)
AWS (c5.4xlarge, 16 vCPUs, 32 GB RAM)	$497
Azure (Standard D16s v5, 16 vCPUs, 64 GiB RAM)	$555
GCP (n2-standard-16, 16 vCPUs, 64 GB RAM)	$431

Table 2: Updated estimated cloud hosting costs (Compute only, On-Demand, Linux, April 2025). Azure instance updated to meet 16 vCPU requirement. Costs vary significantly based on region, OS, usage, and contract type.

6 Conclusions

6.1 Current Version

Given the ambitious scope of VibeRadar, it is important to note that this is a minimum viable product (MVP), and not all the initially envisioned features have been implemented in the current version.

Initially, the project was intended to utilize data from more than five social media sources. However, due to challenges in obtaining academic API keys (e.g., TikTok), high costs (e.g., Instagram), and other complexities and quota limitations (e.g., Twitter), we have focused on the extensive usage of BlueSky and YouTube. However, YouTube also imposes daily usage limitations, which prevent large-scale data ingestion. As a result, the volume of data processed is considerably smaller than originally expected. This is indeed a data volume limitation that we want to artificially suppress by the usage of synthetic data, just to test the performance capabilities of batch ingestion.

Similarly, near-real-time streaming has been proved a success; however, some sources like Youtube did not really offer API streaming capabilities, and do not offer such data volumes to be worth implementing streaming. Nevertheless, it has been implemented, given that data volume is extremely dependent on the topic or query and could yield great results for some products.

Furthermore, while some basic metadata and data governance management have been integrated, a more comprehensive and meticulous approach to metadata management has not yet been implemented.

The project currently includes ingestion metadata and Delta Lake/Spark features for data storage and processing, while additional storage for storing unstructured data (videos, audio) may be added in the second part of the project.

Finally, despite the ongoing nature of the project, it is clear that the immense scope and tight time constraints have made it impossible to fully implement all the core functional features outlined in Section 3. We have prioritized the most relevant features, such as Sentiment Analysis, Real-Time Monitoring, Engagement Tracking, and Predictive Analytics, with the focus on delivering these at most.

On a final note, while there is a desire to implement additional features and complex automation pipelines, the significant time constraints make it virtually impossible to achieve better results within the given limitations.

All in all VibeRadar first launch presents this product as a mature, dynamic and robust solution for impact analysis of commercial retail products. We provide a open-source MVP and a novel system design infrastructure to achieve this goal reproducibly and in any Docker compatible device. It provides state of the art software to achieve a highly capable batch data ETL's to data lake formation and data streaming.

6.2 Evaluation

This section offers a retrospective reflection on the project, discussing both the positive aspects and the challenges encountered, using the evaluation criteria outlined in the project statement. On one hand, the positives were:

- **Dynamicity:** The batch ingestion is extremely flexible through dynamically generated Airflow DAGs from a serialized JSON configuration, utilizing a common "factory pattern" for data pipelines. Adding a new product to track is as simple as modifying a configuration file, and custom use-cases are supported through dedicated DAGs.

- **Reusability:** Thanks to the modular architecture and the abstract factory pattern applied to pipeline generation, the system scales effortlessly to thousands of companies with minimal code changes. This novel approach, termed **"Data Pipelines as Configuration (DPaC)"** coined by the authors, embodies the DevOps mindset and "Everything as Code" paradigm.

- **Openness:** The project was built entirely with open-source technologies, ensuring transparency and reproducibility. Anyone with access to the configuration and a few API keys can replicate the pipelines and results, reinforcing openness and scientific rigor.

- **User-eXperience:** From a developer and data engineer perspective, the UX is streamlined — configurations are simple, modifying or adding pipelines is straightforward, and observability is ensured through Airflow's UI and integrated logging/alerting tools. From an end-user standpoint, intuitive Streamlit dashboards allow for easy access to insights, showcasing the full power of the project through a clean and visually appealing interface.

- **Innovation:** The use of JSON-driven DAG generation combined with the DPaC philosophy introduces a novel, low-maintenance way to manage complex ingestion workflows. It merges automation, modularity, and flexibility in a way rarely seen in traditional data engineering projects. More importantly, the novelty of this product could result in widely industrial usage.

- **Time-to-Insight:** With minimal manual intervention, insights from new products or companies can be delivered rapidly within the same hour since the set-up, accelerating decision-making and data-drivenness.

- **Performance:** Parallel ingestion through Airflow's task scheduling, efficient Kafka streaming for real-time data, and Spark processing for heavy workloads ensure the system performs well under scale. The architecture is clearly designed to be cloud-ready and horizontally scalable.

On the other hand, the negatives were:

- **Data Volume:** A key limitation was not the processing capacity, but rather the restrictive API rate limits imposed by external services (e.g., YouTube API). These limits made it difficult to ingest large amounts of data in a short time, which affected the project's ability to scale ingestion speed for high-frequency or high-volume data sources. Synthethic data will be needed to overcome this shortcoming.

- **Data Quality:** Some data sources inherently include incomplete or noisy data — such as videos with disabled comments, private or removed content, and accounts with limited access. This will affect data consistency and required additional preprocessing and filtering logic on the following steps.

- **Scalability:** Although the architecture was designed with scalability in mind, the current deployment using Docker Compose lacks the elasticity required for horizontal scaling in production environments. Transitioning to orchestration tools like Kubernetes or using managed services would be necessary for a commercial-grade deployment.

7 Future Work and Next Steps

This ambitious and innovative project has laid the groundwork for a modular and scalable data platform for product engagement analytics. So far, the ingestion and landing zones have been successfully implemented, providing a dynamic and reusable data collection layer. The next logical step is to develop the trusted zone, where raw data will undergo validation, cleansing, and standardization to ensure high data quality. This includes enforcing schemas, removing noise, and applying domain-specific rules to produce reliable datasets. Following that, the exploitation zone will be introduced, serving as the interface for business intelligence tools and data consumers, offering curated and query-optimized datasets.

At this stage, defining the final overall data architecture is crucial: whether to continue with the current Data Lake, adopt a 2-tier Data Lake and Data Warehouse structure, or transition to a Data Lakehouse that combines both. Additionally, implementing data lineage and metadata tracking will be essential for ensuring traceability and compliance, while operational excellence will require real-time monitoring and alerting for pipeline health.

To enhance scalability and fault tolerance, migrating from Docker Compose to Kubernetes or another production-grade orchestration platform is a priority. Introducing CI/CD pipelines will streamline deployment and configuration management.

Finally, synthetic data management will be a key point of improvement in order to enable the overall project performance stress-testing.