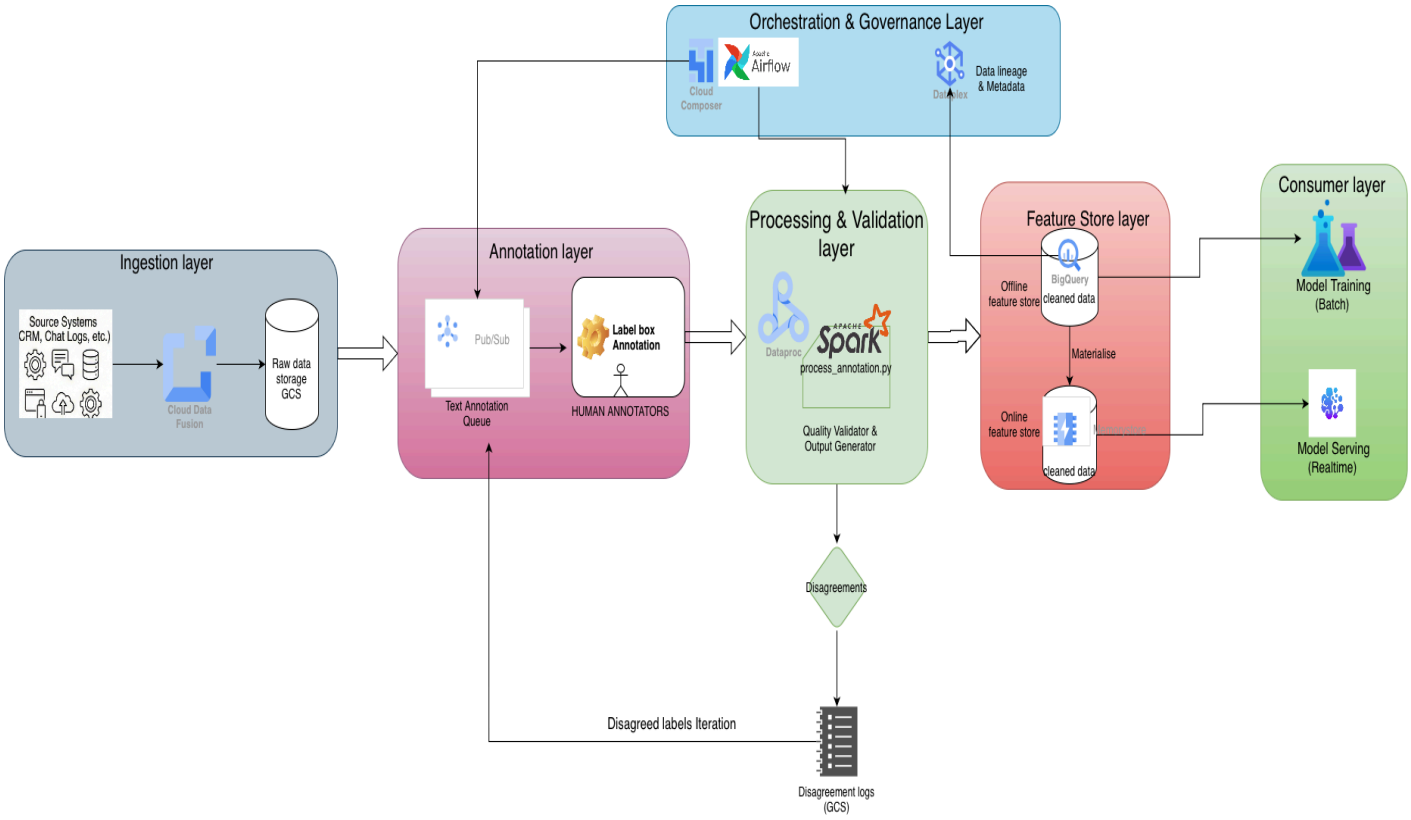
1.) Architectural Diagram



2.) Components Description

Ingestion Service: Cloud Data Fusion

High-level, managed ETL/ELT Ingestion service with built-in connectors for databases, files and SaaS apps, reducing the need to write complex connector code.

Annotation Queue: Pub/Sub

Serverless, fully managed, high-throughput message queue native to the GCP ecosystem. Ensures decoupling and reliable delivery of text samples for annotation and labels back to the system. This service is used to continuously stream the data as annotations are being labeled to Apache spark for processing from annotation platform such as label box, label studio etc.

Orchestration: Apache Airflow (via Cloud Composer)

Industry standard for managing complex DAGs. Composer provides a managed Airflow environment on GCP, simplifying scaling and maintenance. Ideal for managing job dependencies (e.g., ensuring feature engineering starts only after ingestion is complete). Also provides the Monitoring of jobs, Scheduling and Backfill strategy.

Processing Engine: Apache Spark (via Dataproc)

Provides distributed, scalable compute power essential for feature engineering, running quality validation checks across massive datasets, and performing backfills. Dataproc is the managed Spark service on GCP.

Offline Feature Store: Bigquery

BigQuery is a fully serverless managed Google Data warehouse, It has native time travel, which is crucial for point-in-time correctness. BigQuery is excellent for large-scale analysis with high performance and scalability.

Online Feature Store: MemoryStore

Extremely fast, in-memory key-value store providing the **sub-millisecond latency** required for real-time model serving lookups. Memorystore is the managed Redis/Memcached service on GCP.

Data lake: Google Cloud Storage

This serves as the foundation of your Data Lake architecture. It is the central, highly durable, and cost-effective storage layer for all raw data. GCS is used here to store the outputs of the **Quality Validator**, specifically the files containing the records that failed quality checks or

agreement checks. This data is kept separate from the clean training set for auditing and for the Disagreed Labels Iteration loop.

3. Technology Justification & Trade-offs

Here is the deep-dive justification for each choice, including why it was selected over alternatives and the specific trade-offs accepted.

A. Ingestion: Cloud Data Fusion

- Why Best: It provides a visual, low-code interface with 150+ pre-built connectors. This drastically reduces the "time-to-first-byte" compared to writing custom Python/Airflow ETL scripts.
- Trade-off:
    - *Pros:* Rapid development, native GCP integration, lower maintenance.
    - *Cons (Trade-off):* Less flexibility than custom code for highly complex, non-standard transformations. Can be more expensive than a simple script for very small data volumes.

B. Annotation Queue: Cloud Pub/Sub

- Why Best: It allows the system to buffer incoming text requests asynchronously. If the annotation team sleeps or the Labelbox API goes down, Pub/Sub holds the messages safely.
- Trade-off:
    - *Pros:* Serverless (no cluster management), global scalability, integrates natively with Cloud Functions/Dataflow.
    - *Cons (Trade-off):* Unlike Apache Kafka, Pub/Sub has limited "replay" capabilities (retention is typically shorter and seek operations are less granular). Kafka offers better long-term storage but requires heavy operational management.

C. Processing: Dataproc (Spark)

- Why Best: Essential for "backfilling" years of historical data. Standard Python scripts cannot handle terabytes of text logs efficiently. Spark distributes this load across a cluster.
- Trade-off:
    - *Pros:* Massive horizontal scalability, rich ecosystem for ML (Spark MLlib).
    - *Cons (Trade-off):* Cluster spin-up time can add latency (minutes) compared to stream processing (Dataflow). Overkill for small datasets (<10GB).

D. Offline Store: BigQuery (with Time Travel)

- Why Best: Supports standard SQL for data scientists and provides Point-in-Time Correctness via native history or Delta Lake formats. This ensures you can retrain a model on "data as of last month" exactly.
- Trade-off:
    - *Pros:* Serverless, separates storage from compute, extremely fast for aggregations.
    - *Cons (Trade-off):* Higher latency for single-row lookups (not suitable for online serving). Costs can spike if queries are not optimized (scans full tables).

E. Online Store: Memorystore (Redis)

- Why Best: Models need features in <10ms to respond to a user. BigQuery takes seconds. Memorystore provides the sub-millisecond read speeds required for real-time inference.
- Trade-off:
    - *Pros:* Blazing fast, industry standard (Redis).
    - *Cons (Trade-off):* RAM is expensive. Data is volatile (if persistence isn't configured correctly). It does not hold historical data, only the "latest" state.

---

4. Data Governance Implementation

To ensure the pipeline is auditable and reproducible, the following governance strategies are implemented:

1. Data Lineage (Source → Feature):
    - Every record moving through the pipeline is tagged with a metadata envelope containing the `Raw_Data_GCS_Path` and the `Airflow_DAG_Run_ID`.
    - Result: If a model fails, you can trace the feature vector back to the exact source file and the specific processing job that created it.
2. Code Versioning:
    - Transformation logic (feature engineering code) is versioned in Git. Airflow passes the specific `Git Tag` (e.g., `v1.2.0`) to the Dataproc job.
    - Result: You know exactly *which version of the code* generated a specific dataset.
3. Label Versioning:
    - Labels are stored with a `Label Creation Timestamp`. The Feature Store uses this timestamp to filter data during training generation, preventing "data leakage" from the future.