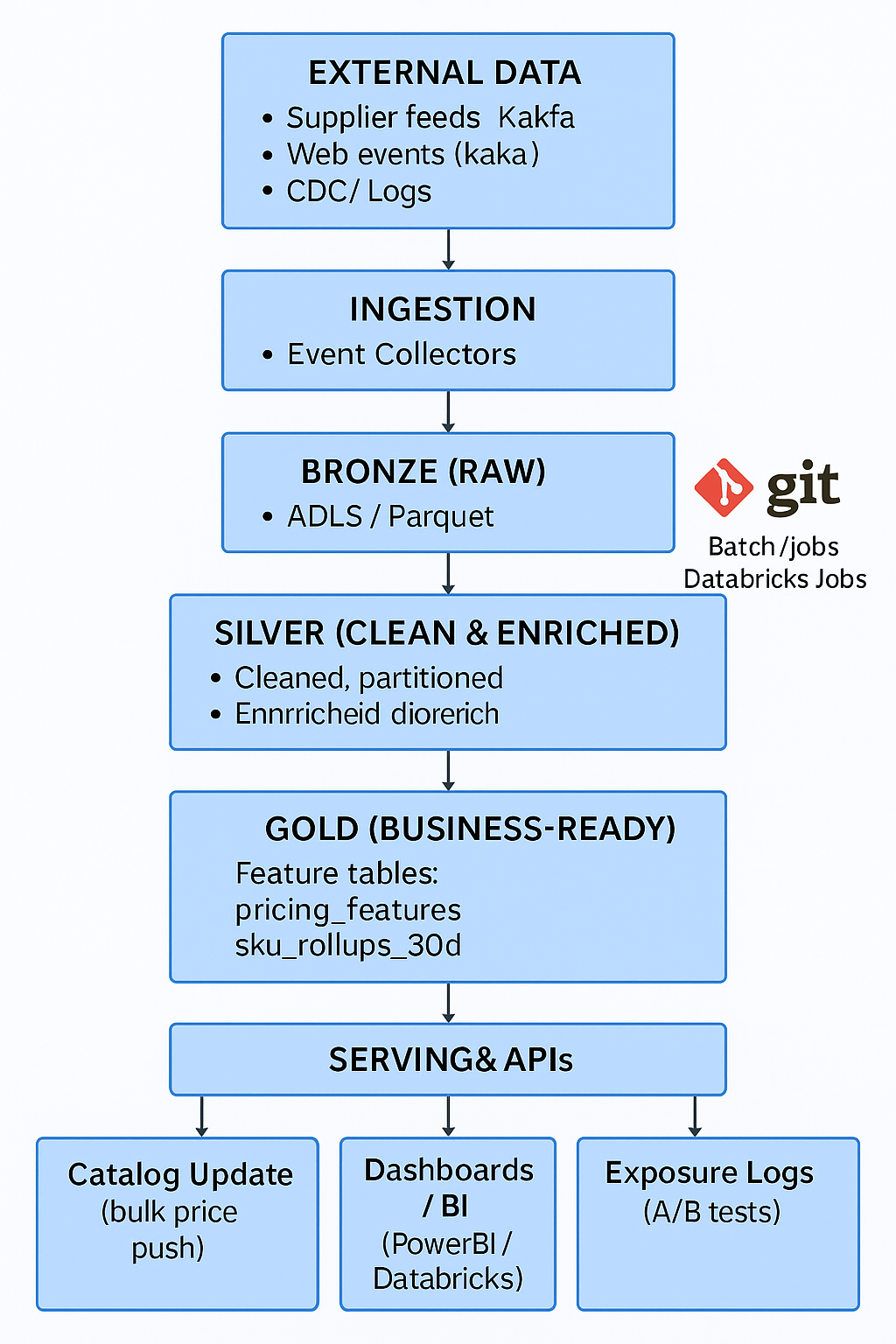
### **High-level goals**

* Increase expected margin while protecting conversion.
* Keep anchor SKUs competitively priced to preserve basket pull.
* Provide auditable, explainable price suggestions (important for medical supplies).

**Architecture Diagram**

****

**Ingestion layer for batch and/or streaming data**

**Ingestion**

* Scheduled batch runs (Airflow tasks) to land product, supplier, transaction, clickstream and competitor feeds into Raw Lake (Parquet) (ADLS)
* File Based triggering of airflow tasks for required cases

**Storage and Partitioning strategy :**

* **Bronze Layer** (ADLS data) -> **Silver layer** (store data into delta lake  
  after applying partitioning techniques and filtering of unnecessary data) -> **Gold layer** (ADLS with CSV or parquet formats based on requirement)
  + Bronze Layer - Copy raw data as is from source and store that into ADLS container so that we can utilize it for later improvisation. Based on nature of data we can decide whether to overwrite entire data or appending new deltas
  + Silver Layer - Filtering of unnecessary columns and applying partitioning on columns which have higher usage of filtering.

To decide partitioning techniques we can decide based on type of data (dimension or transactional data) like partitioning on **category for product data (dimension data)** , **partitioning on region / transaction years for transactional data** . If we are working with different regions world wide it will be beneficial to partition on regions so we can fetch region specific data and work on it .We will have transactional data from years. So if we partition it, it will be useful to select data based on the requirement of timeline for model training . Apply all transformations here based on Dynamic pricing strategies

* + Gold Layer - Store final aggregated data back into ADLS in required formats (either CSV or parquet  
    We can connect this ADLS data to either PowerBI dashboards (for analytics purpose) or we can feed this data to ML models ( for Machine learning purpose). For building models we can utilize Databricks ML

**Feature Engineering and transformation logic:**

After ingestion and cleaning, we transform raw product, transaction, and competitor data into derived features that support rule-based pricing. Key transformations include:

* Anchor identification: Mark anchor SKUs based on user consumption patterns (e.g., top sellers or high conversion rates). Anchors are then kept competitive by comparing against rival prices and applying reductions if necessary, while non-anchor products may allow for higher margins.
* Cost margin enforcement: Enforce a minimum price floor to avoid unprofitable sales. Margin is defined as (selling\_price – cost\_price) / cost\_price, and suggested prices always respect this threshold.
* Conversion history adjustment: Incorporate historical purchase behavior by calculating conversion\_rate = purchases ÷ clicks. If conversions remain stable at higher prices, competitive adjustments are not required. If conversions decline, the pricing logic shifts toward competitive levels.
* Category-specific rules: Apply different pricing strategies for different product families. Consumables typically operate at lower margins but higher volumes, while instruments and specialized equipment justify higher margins. Category-aware rules such as margin bands or rounding conventions ensure consistency and fairness

### **Experimentation Support :**

To safely evaluate new pricing strategies, the platform incorporates built-in A/B testing and event logging.

* A/B toggles: Customers are randomly assigned to either a baseline policy (current pricing) or a new pricing policy (suggested prices). Assignments are recorded in an exposure table with fields such as experiment\_id, sku\_id, user\_id (or cohort), and assigned\_policy. The pricing engine reads these assignments to ensure consistent treatment for each customer during the experiment.
* Logging: Each time a user is shown a price, the system records an exposure event including the assigned policy, suggested price, reason code (e.g., “anchor\_competitive” or “profit\_optimized”), and timestamp. When an order or return occurs, an outcome event is logged and linked to the original exposure via exposure\_id. This linkage enables unbiased comparison of conversion rates, margins, and revenue between baseline and test groups.

By combining toggles with detailed exposure–outcome logging, the system provides a robust framework for experimentation, ensuring that pricing adjustments are validated with real customer behavior before wider rollout

### **Serving Layer for Real-Time or Scheduled Access to Outputs :**

The serving layer makes pricing outputs available to downstream systems in two modes:

* Scheduled batch access: A nightly Databricks Job or Airflow task writes suggested prices into Delta/Parquet tables. These outputs can be consumed by dashboards, reporting tools, or batch catalog updates.
* Real-time access: An optional lightweight API (e.g., FastAPI or Flask) retrieves the latest pricing features and returns suggested prices for specific SKUs on demand

### **Where and How Databricks is Integrated :**

Databricks is used for ETL pipelines, data transformations, and storage in Delta Lake. Notebooks provide an interactive environment for prototyping, while Databricks Jobs schedule recurring data processing tasks. Delta Lake’s time-travel and ACID capabilities ensure reliable data management.

### **Reliability and Scalability Considerations :**

Partitioned Delta tables, incremental processing, and cluster auto-scaling ensure the solution can handle growing transaction volumes. Airflow or Databricks Jobs provide retries and monitoring for reliability.

### **Where AI-Assisted Development Tools Were Leveraged :**

AI tools such as ChatGPT supported understanding requirements, Designing basic skeleton design, and drafting system design text. Debugging. They accelerated development. Enhancing comments in code and naming conventions and producing data.

**What We Would Do Differently with More Time :**

**End to End pipeline building from ADLS (raw data) and scheduled airflow tasks to fetch data from different sources into ADLS . ETL pipelines in databricks. Integration of GIT with Databricks. More research on what else we can include to make a better pricing system like below topics.**

With more time, we could make pricing smarter by looking at additional factors. For example, we could adjust prices based on **user behavior** (repeat vs. new customers), **region** (local demand and supply), or **mode of payment** (e.g., offer discounts for online payments). We could also include **seasonality trends** (higher demand during certain months) and focus on **highly sold products** to refine anchor-product rules. These improvements would help us personalize pricing better and capture more business opportunities.

### **Trade-offs and Reasoning :**

We designed the data platform using a Bronze–Silver–Gold layered architecture on Delta Lake for clarity and governance. The trade-off is that this approach introduces additional storage costs and some latency compared to a single-layer pipeline. However, the benefit is strong data quality control, reproducibility through time-travel, and easier debugging when issues arise. The added reliability and transparency outweigh the cost and latency overhead for our use case.