SCALABILITY ARCHITECTURE

Scaling the Retail Insights Assistant

Architecture Design for 100GB+ Datasets

From in-memory Pandas to distributed cloud-native analytics

February 2026

SLIDE 1

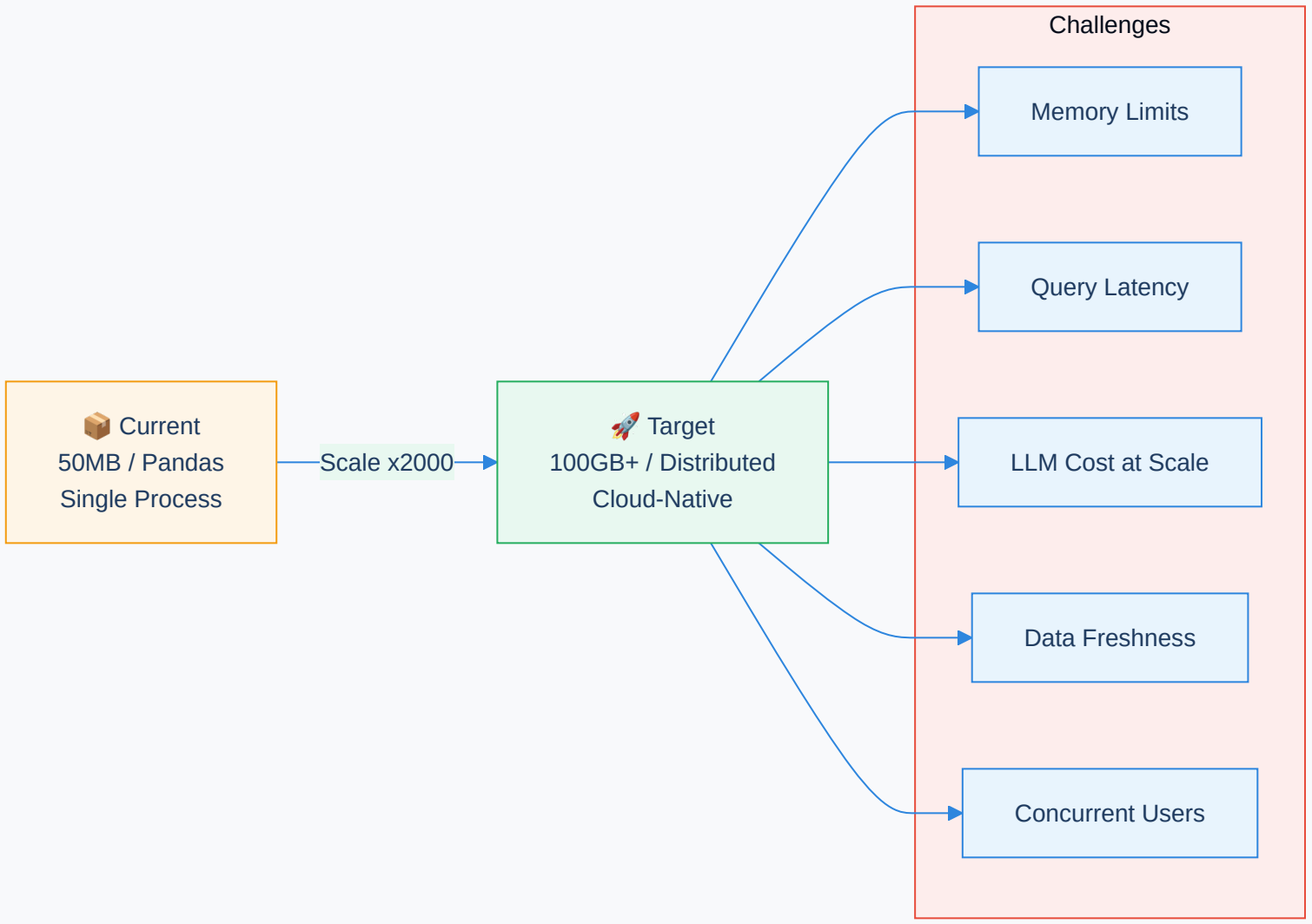
Current State vs. Scale Target

📦 Current Demo (~50MB)

- Single-process Streamlit app
- All data loaded into Pandas in-memory
- 7 CSV files, ~175K total rows
- 18 deterministic Pandas tools
- 3 sequential LLM calls per query
- No caching or persistence layer

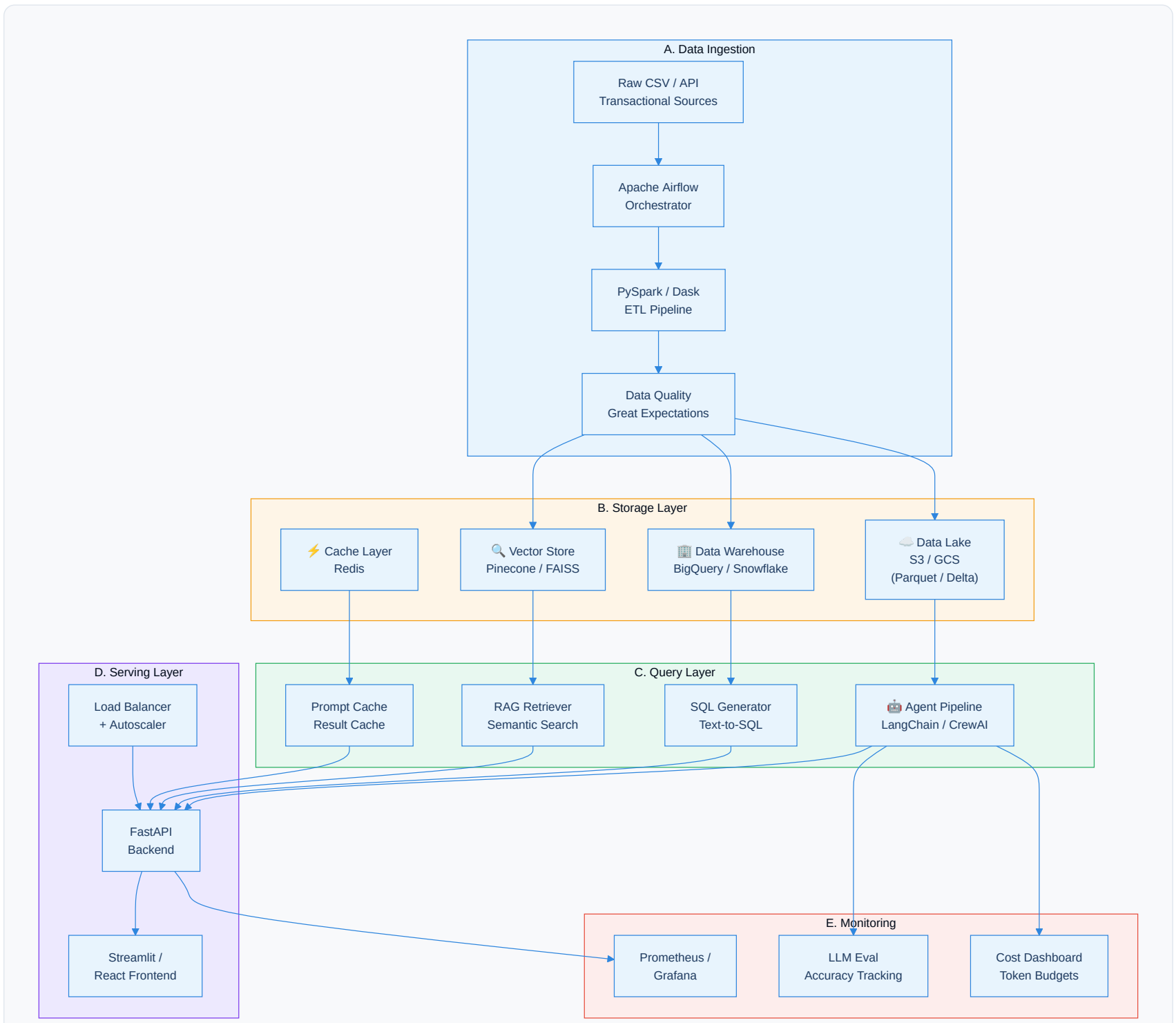
🚀 Target Scale (100GB+)

- Distributed compute (PySpark / Dask)
- Cloud data warehouse (BigQuery / Snowflake)
- Data lake on object storage (S3 / GCS)
- RAG with vector embeddings (FAISS / Pinecone)
- Prompt caching & query result caching
- Monitoring, cost control, autoscaling



SLIDE 2

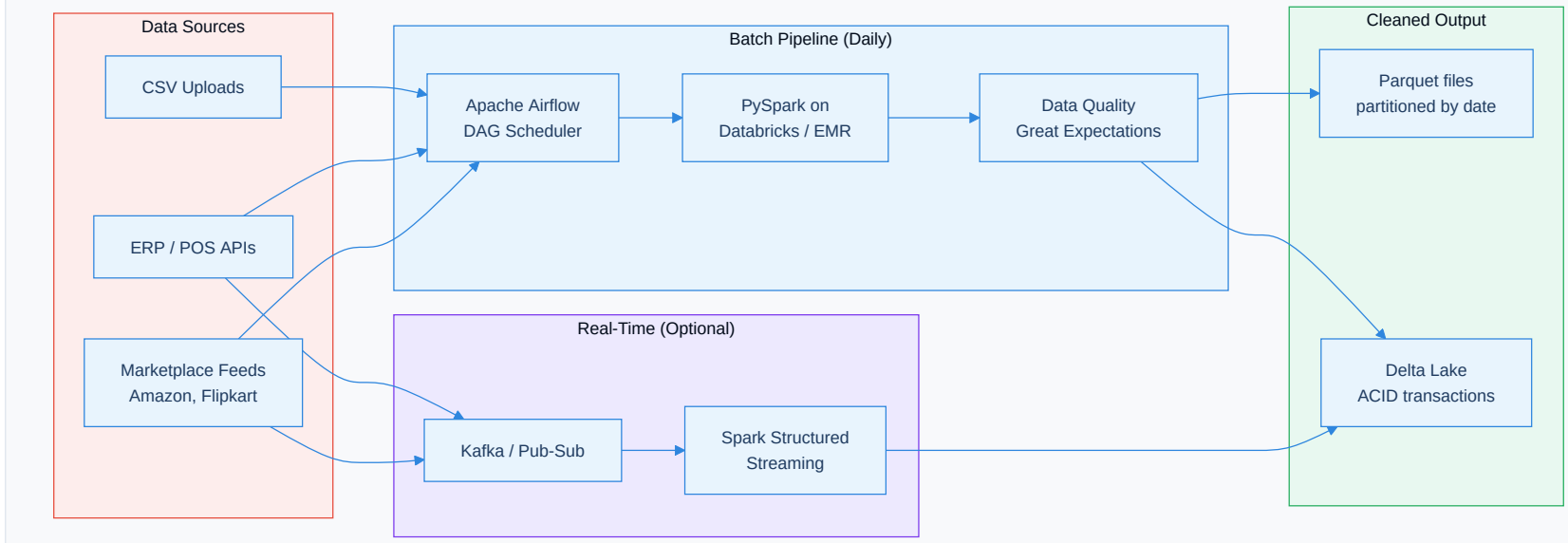
High-Level Scaled Architecture



Design Philosophy: The architecture separates concerns into five layers — Ingest, Store, Query, Serve, Monitor — each independently scalable. The agent pipeline remains the orchestration brain, but data operations move from in-memory Pandas to distributed SQL and vector search.

SLIDE 3

A. Data Engineering & Preprocessing



Batch Processing Strategy

PySpark on Databricks

- Distributed DataFrame operations across cluster
- Handles 100GB+ CSV ingestion in minutes
- Schema enforcement and type coercion at scale
- Partitioned writes by `date` and `category`

Data Quality Gates

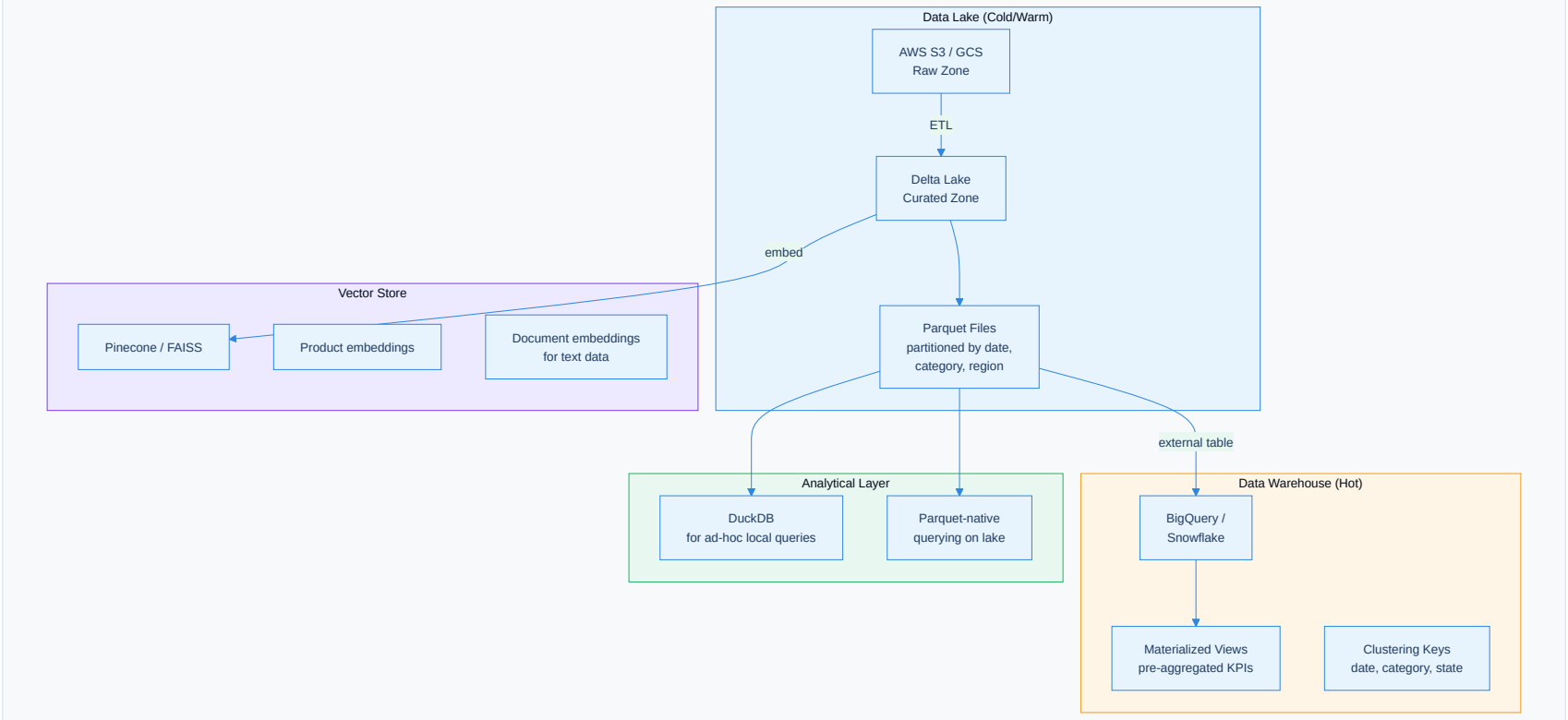
- **Great Expectations** for validation rules
- Null checks, range validation, referential integrity
- Schema drift detection across ingestion runs
- Quarantine bad records; alert on threshold breaches

Cleaning Pipeline at Scale

Step	Current (Pandas)	Scaled (PySpark)
Column normalization	`df.columns.str.replace()`	`df.toDF(*normalized_names)`
Date parsing	`pd.to_datetime()`	`F.to_timestamp()` with format
Numeric coercion	`pd.to_numeric(errors='coerce')`	`F.col().cast(DoubleType())`
Status flags	Vectorized `.isin()`	`F.when().otherwise()` UDF
Output format	In-memory DataFrame	Partitioned Parquet / Delta Lake

SLIDE 4

B. Storage & Indexing



Storage Tier Strategy

Tier	Technology	Data	Access Pattern	Cost
Cold	S3 / GCS (Standard IA)	Raw CSVs, archives	Infrequent; batch ETL	~$0.01/GB/mo
Warm	Delta Lake on S3/ GCS	Curated Parquet, versioned	ETL output; analytical queries	~$0.02/GB/mo
Hot	BigQuery / Snowflake	Materialized views, KPI tables	Low-latency agent queries	~$5/TB scanned
Vector	Pinecone / FAISS	Embeddings (products, documents)	Semantic similarity search	~$70/mo (Pinecone Starter)

Indexing Strategies

Columnar Partitioning

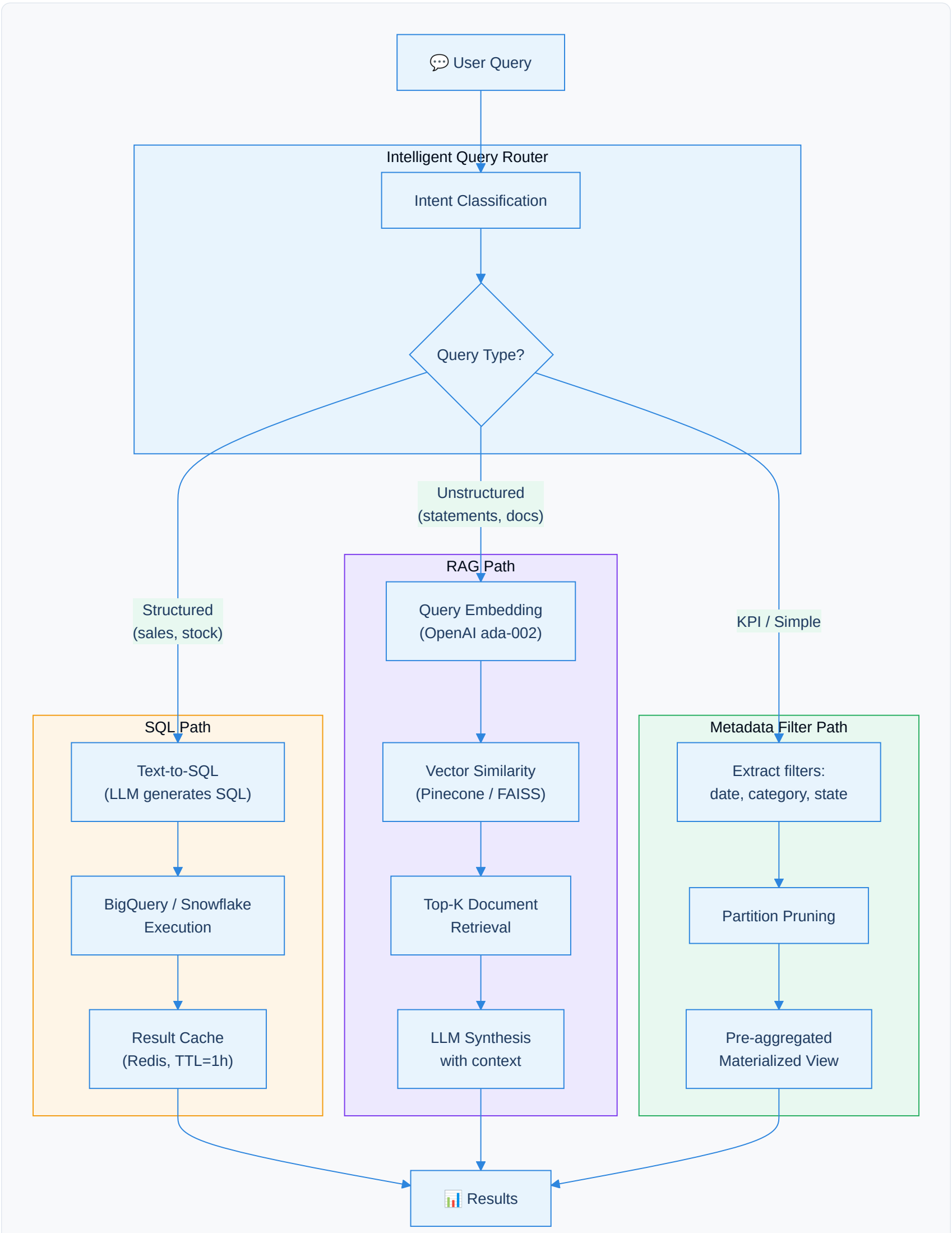
- Partition by `year/month/day` for time-range queries
- Cluster by `category`, `state` for filtered aggregations
- Parquet columnar format: only read needed columns
- Predicate pushdown eliminates full scans

Materialized Views

- Pre-computed KPIs: total sales, cancellation rate, stock levels
- Refreshed daily by Airflow pipeline
- Sub-second query response for common questions
- Reduces warehouse compute costs by 80%+

SLIDE 5

C. Retrieval & Query Efficiency



Three Retrieval Strategies

1. SQL Generation (Structured Data)

For questions about sales, inventory, pricing — the LLM generates SQL that runs directly on BigQuery/ Snowflake.

2. RAG (Unstructured Text)

For expense statements, warehouse comparisons, and document-type queries.

- Text-to-SQL with schema context injection
- Query validation before execution
- Result caching in Redis (TTL-based)
- Row limit enforcement (max 100 rows returned)

- Documents chunked (512 tokens) and embedded
- Query embedded → top-K similarity search
- Retrieved chunks injected as LLM context
- Handles new document types without code changes

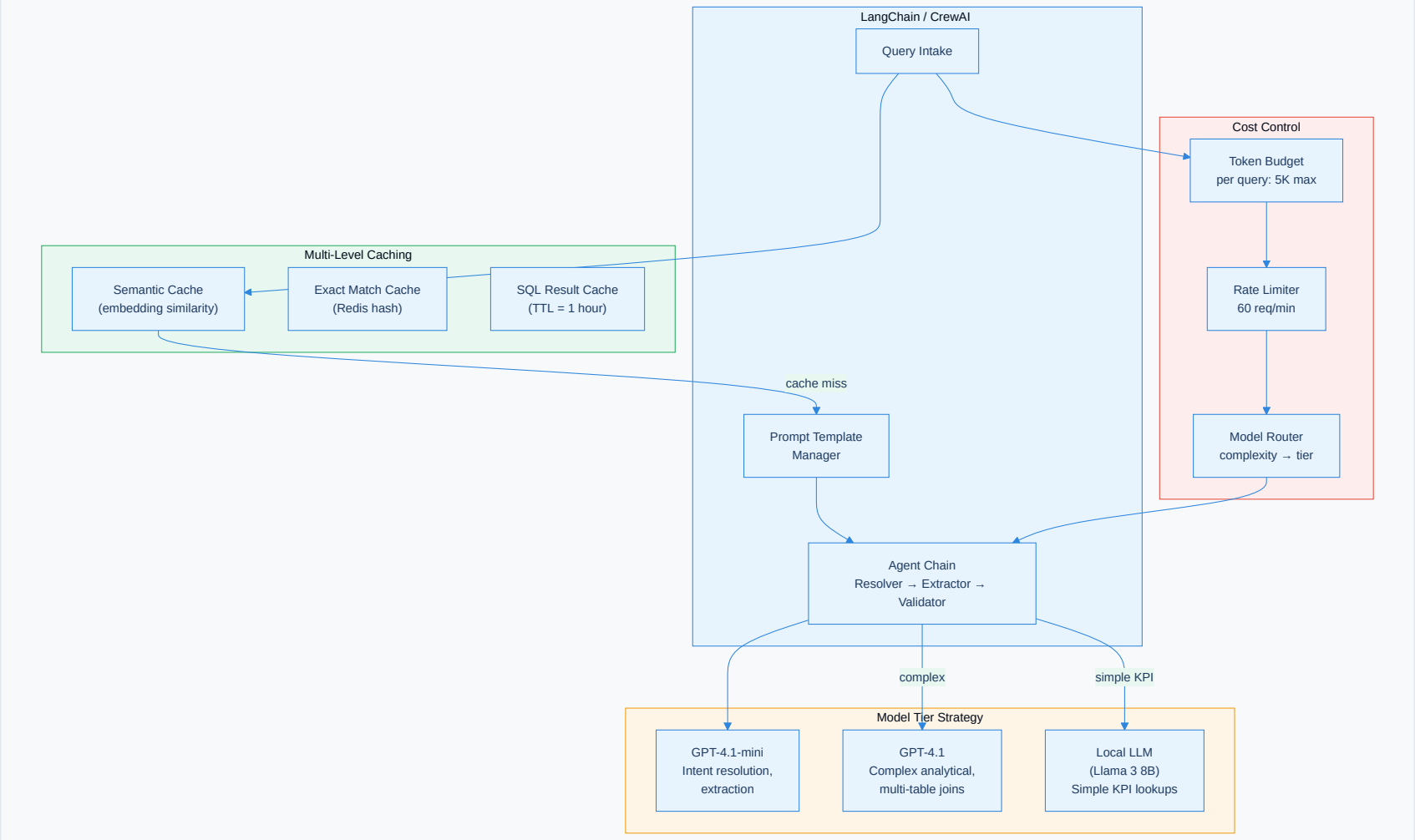
3. Metadata Filtering + Materialized Views (KPI Queries)

Simple KPI queries ("What is the cancellation rate?") bypass LLM SQL generation entirely and hit pre-computed materialized views, delivering sub-second responses.

Key Optimization: The Resolver Agent classifies query type first, routing to the cheapest and fastest retrieval path. Only complex analytical queries incur full SQL generation + warehouse scan costs.

SLIDE 6

D. Model Orchestration at Scale



Prompt Management

Template Versioning

- Prompts stored as versioned templates
- A/B testing of prompt variants
- Schema context auto-injected from warehouse metadata
- Few-shot examples managed in a prompt registry

Chain Optimization

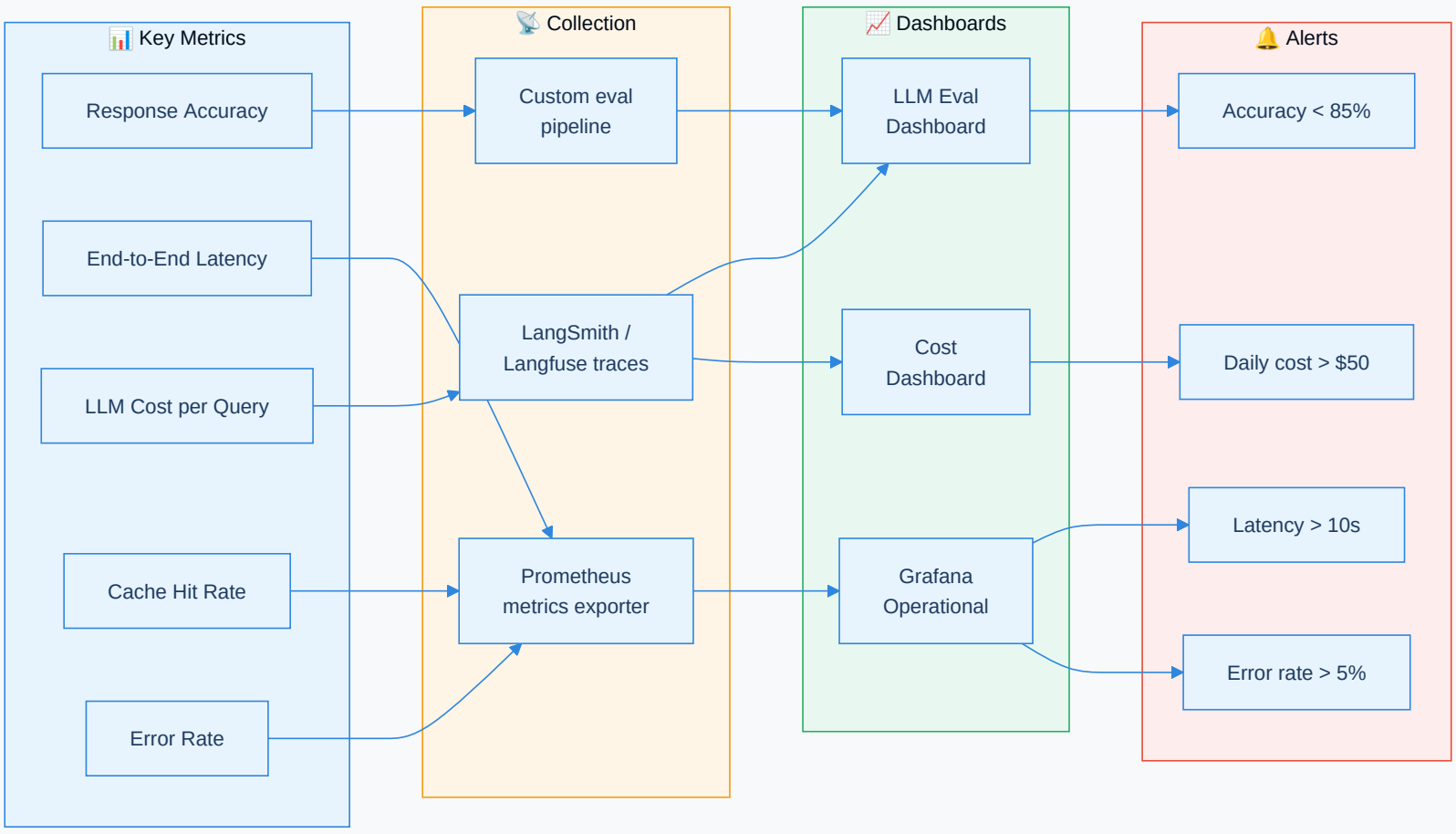
- Parallel agent execution where possible (Extractor + Validator batched)
- Short-circuit: skip Validator for cached / high-confidence answers
- Streaming responses for better UX latency
- LangChain callbacks for trace logging

Cost Optimization

Strategy	Mechanism	Estimated Savings
Semantic caching	Cache responses for similar queries (cosine > 0.95)	40–60% fewer LLM calls
Model tiering	Use cheaper models for simple queries	30–50% cost reduction
Token budgets	Truncate context, limit output tokens	20% per-call savings
Materialized views	Bypass LLM entirely for pre-computed KPIs	100% savings for KPI queries
Batch processing	Queue and batch similar queries together	15–25% throughput improvement

SLIDE 7

E. Monitoring & Evaluation



Evaluation Framework

Accuracy Metrics

- **Factual correctness:** Compare LLM answers to ground-truth SQL results
- **Numerical precision:** Automated check for ₹ amounts within 1% tolerance
- **Completeness:** Did the answer address all parts of the question?
- **Hallucination detection:** Flag claims not supported by tool results

Operational Metrics

- **P50 / P95 / P99 latency:** Target P95 < 8s
- **Token usage:** Per-agent and per-query tracking
- **Cache hit rate:** Target > 40% for repeat patterns
- **Tool execution time:** BigQuery scan duration tracking

Error Handling & Fallback Strategies

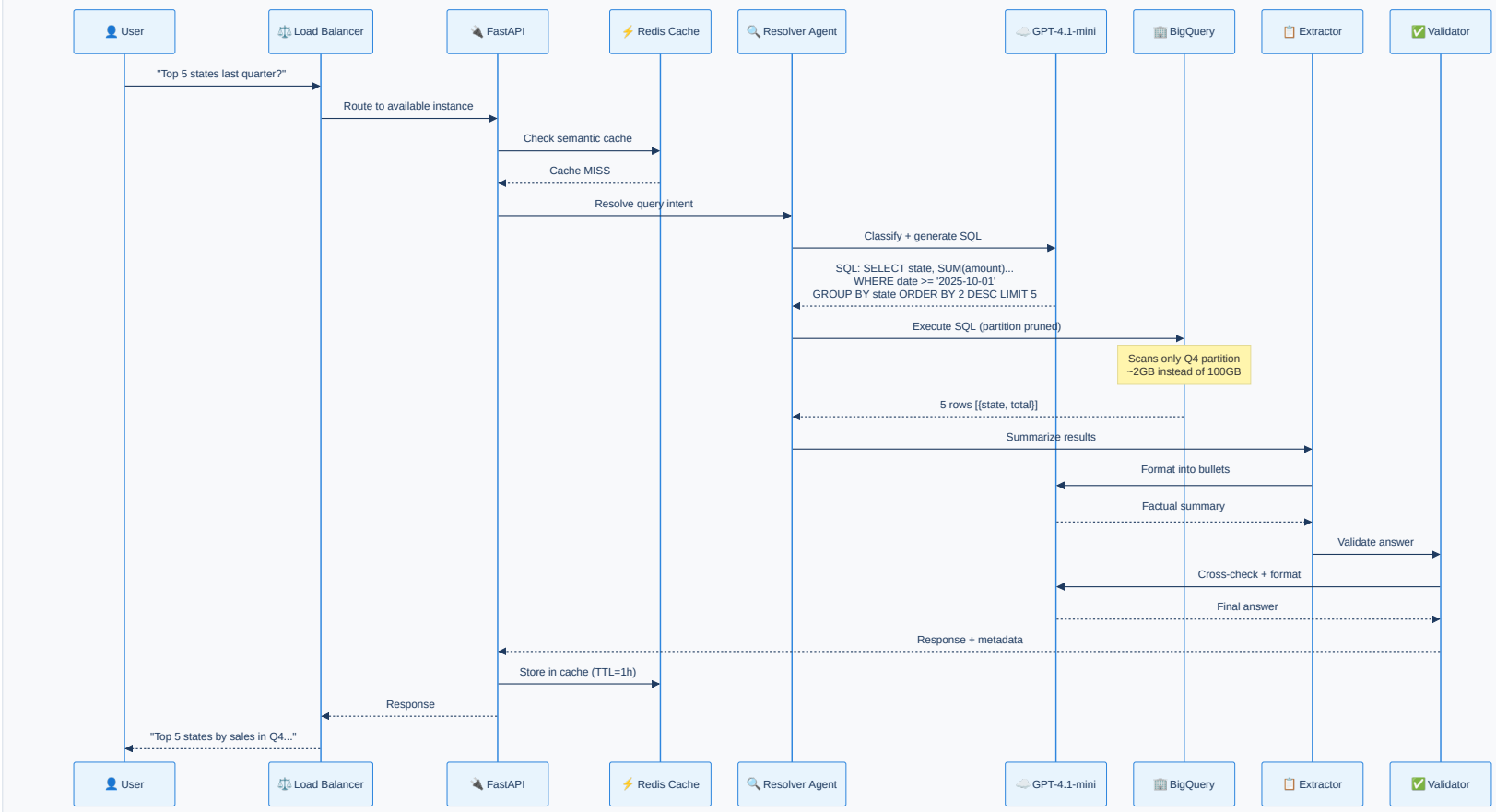
Failure Mode	Detection	Fallback Strategy
LLM returns invalid JSON	JSON parse failure	Heuristic planner (keyword matching)
LLM hallucination	Answer not grounded in tool results	Return raw data with disclaimer
SQL generation error	BigQuery execution exception	Retry with simplified prompt; fallback to materialized view
High latency (>15s)	Timeout monitoring	Cancel and return cached similar answer
LLM API outage	HTTP 5xx / timeout	Switch to backup model provider; queue request
Low confidence response	Validator flags uncertainty	Return answer with confidence disclaimer; log for human review

Human-in-the-Loop: Queries flagged as low-confidence are queued for human review. Reviewed answers feed back into the few-shot prompt registry, continuously improving accuracy.

SLIDE 8

Scaled Query-Response Pipeline

> **Example Query:** *"Which 5 states generated the highest sales last quarter?"* on a 100GB+ dataset



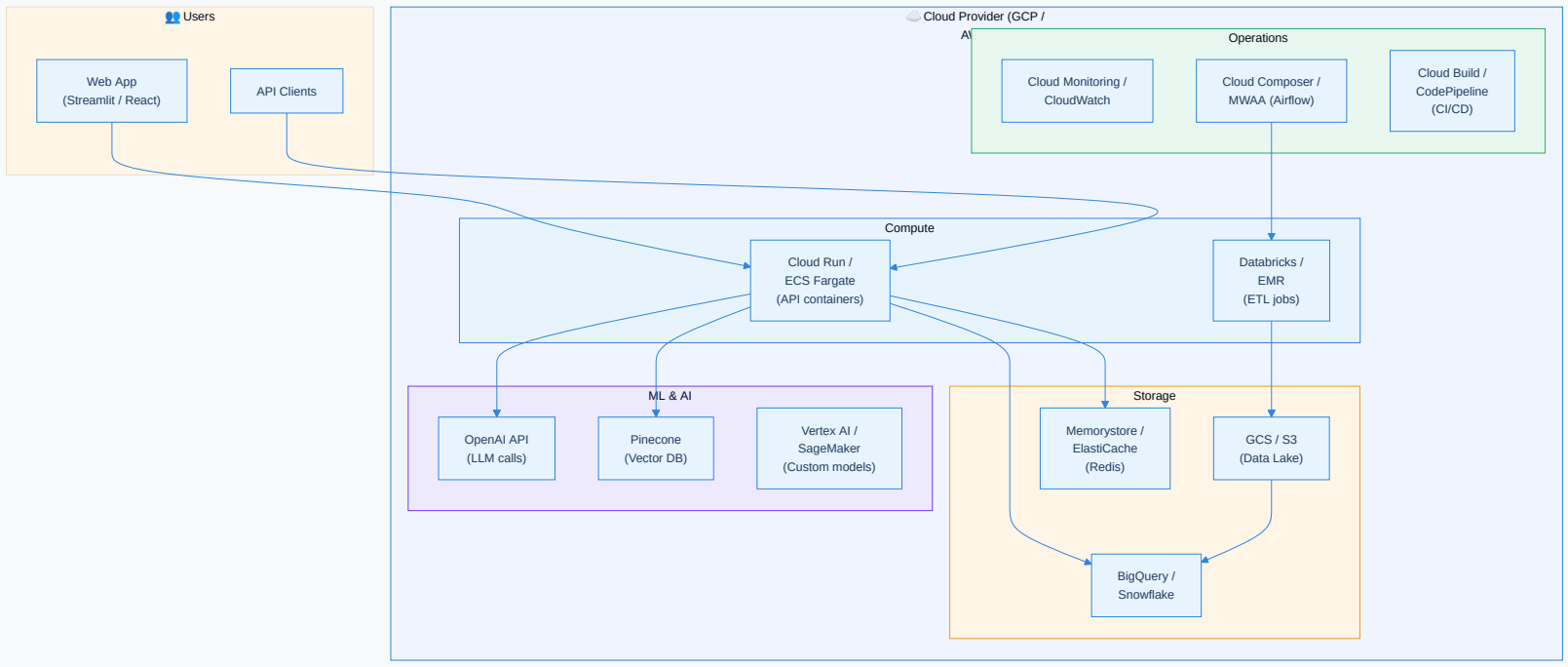
Performance at Scale

Stage	Current (50MB)	Scaled (100GB+)
Data load	2–3s (one-time)	0s (pre-loaded in warehouse)
Query resolution	1–2s (LLM)	1–2s (LLM, same)
Data retrieval	<100ms (Pandas)	1–3s (BigQuery, partition-pruned)
Extraction + Validation	2–4s (2 LLM calls)	2–4s (2 LLM calls, same)
Total (cold)	**4–8s**	**4–9s**
Total (cache hit)	N/A	**<200ms**

> **Key Insight:** With partition pruning and materialized views, the 100GB+ system achieves comparable latency to the current 50MB demo — the warehouse scans only the relevant data slice, and caching eliminates repeat queries entirely.

SLIDE 9

Infrastructure & Deployment

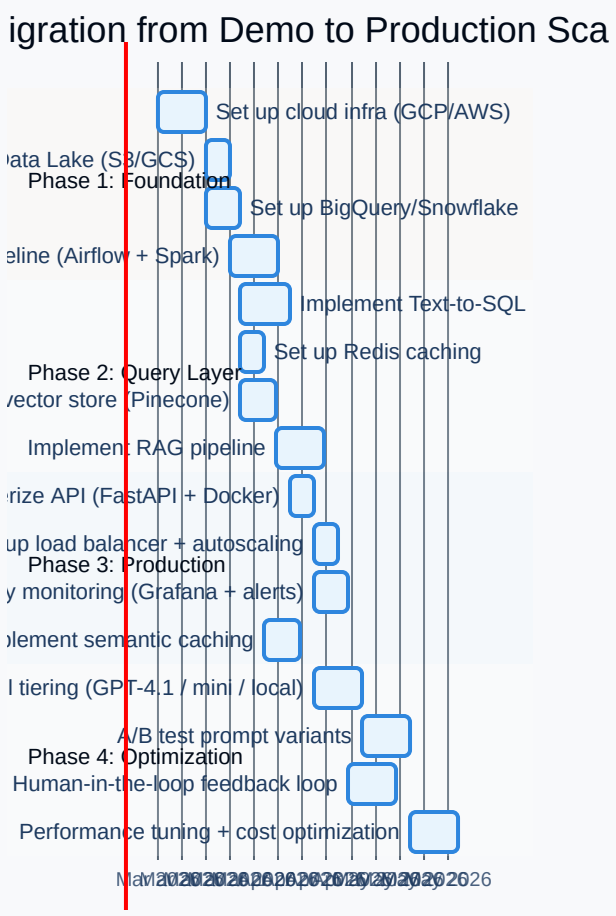


Estimated Monthly Costs (100GB Dataset, 1K queries/day)

Component	Service	Est. Monthly Cost
Compute (API)	Cloud Run / Fargate (2 instances)	$50–100
Data Warehouse	BigQuery on-demand	$100–300
Data Lake	GCS / S3 (100GB)	$2–5
LLM API	OpenAI GPT-4.1-mini (3K queries × 3 calls)	$60–180
Cache	Redis (Memorystore / ElastiCache)	$30–60
Vector DB	Pinecone Starter	$70
ETL Orchestration	Cloud Composer / MWAA	$100–200
Total		**$400–900/mo**

SLIDE 10

Migration Roadmap



Timeline: Estimated 10–12 weeks from demo to production-scale deployment, with Phase 1 (foundation) completing in the first 4 weeks and immediate ROI from the query layer in Phase 2.

Key Migration Principles

Incremental Migration

Each phase delivers standalone value. The system remains functional throughout — new capabilities layer on top of existing ones.

Backward Compatible

The 3-agent architecture (Resolver → Extractor → Validator) is preserved. Only the underlying data access and caching layers change.

Cost-Aware

Each phase includes cost monitoring. Scaling decisions are driven by actual usage metrics, not upfront provisioning.