Hybrid Symbolic–Semantic Multimodal RAG for Financial KPI Reconstruction from PDFs

Abstract

Context: Financial reports are inherently *unstructured, fragmented and very noisy*, combining not only text but tables, images, and graphical representations across multiple pages.

Problem: Standard retrieval pipelines fail to reconstruct Key Performance Indicators (KPIs) that require **cross-page and cross-document dependency resolution** and **semantic + structural reasoning**.

Limitation of Existing Methods:

- OCR or pdf parsing → brittle and non-generalizable
- Standard RAG → lacks dependency modeling
- Embeddings → no notion of *compositional completeness*

Proposed Contribution: We introduce a **hybrid symbolic–semantic multimodal architecture** combining:

- Vision-based structured extraction
- Signed vector symbolic retrieval space
- Hybrid scoring function (semantic + deterministic)
- LLM-guided dependency graph resolution

Impact: The system reconstructs **validated, auditable KPI datasets** with:

- dependency tracking
- mathematical consistency guarantees
- full observability and traceability

1 Introduction

1.1 Problem Definition

Financial KPI extraction from PDF documents is fundamentally a **non-trivial reconstruction problem**:

$$\text{PDF} \rightarrow \{\text{text}, \text{tables}, \text{graphs}\} \rightarrow \text{KPI}$$

This transformation is ill-posed due to:

- Multi-modal representation (text + tables + charts + graphs)
- Multi-page and multi-document(past periods) dependencies
- Inconsistent KPI and it's variables naming conventions

Key Insight: A KPI is not always explicitly stated. It is often **distributed across multiple fragments** that must be recomposed.

1.2 Why Standard Approaches Fail

(1) OCR + Rule-Based Systems

$$\text{PDF} \rightarrow \text{OCR} \rightarrow \text{Rules}$$

- High sensitivity to layout variation
- No semantic understanding
- No robustness to unseen formats

(2) Standard RAG Pipelines

$$\text{chunk} \rightarrow \text{embedding} \rightarrow \text{retrieve}$$

Failure modes:

- KPI components split across chunks
- No structural reasoning
- No dependency graph modeling

Critical Limitation: Standard RAG assumes *answer locality*, while KPI reconstruction requires **distributed reasoning**.

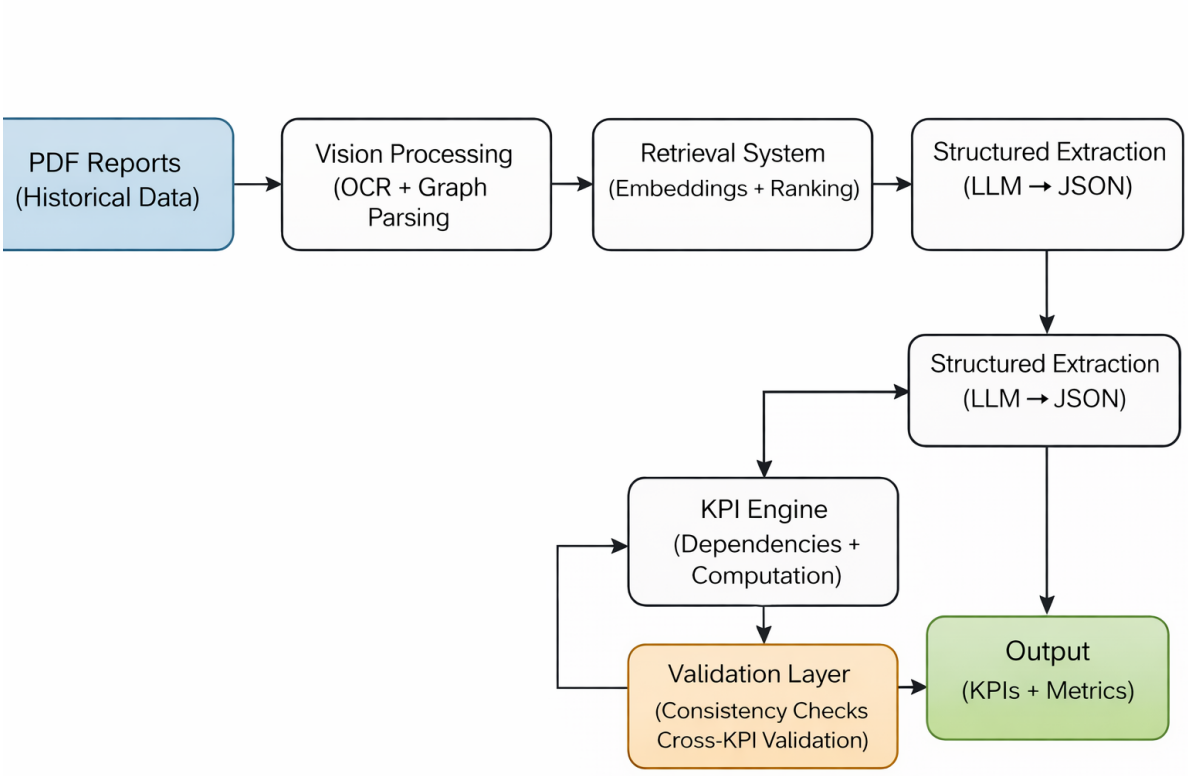
1.3 Core Contributions

- Multimodal extraction (Vision + Text)
- Signed vector symbolic retrieval space
- Hybrid scoring function:

$$f = w_{\text{meta}} + \lambda \cdot \cos(\text{similarity})$$

- LLM-based dependency graph construction
- Deterministic validation and observability layer

System Overview



Interpretation: The architecture decomposes the pipeline into:

- Extraction Layer (Vision + Metadata)
- Retrieval Layer (Symbolic + Semantic)
- Reconstruction Layer (LLM + Math Engine)

2 Formalization of the Retrieval Problem

We define a KPI as a function:

$$K = f(x_1, x_2, ..., x_n)$$

where each x_i may be:

- Explicitly observed
- Implicit (requires retrieval)
- Computable

We introduce a symbolic representation:

$$v \in \{-1, 0, 1, \text{None}\}^d$$

Interpretation:

- $0 \rightarrow$ direct KPI match
- $\pm 1 \rightarrow$ complementary components
- None $\rightarrow$ absence

Key Innovation: Retrieval becomes a **constraint satisfaction problem** instead of similarity search.

Why This System is Fundamentally Different

- Not a simple RAG → a **computation-aware retrieval system**
- Not just embeddings → **symbolic algebraic space**
- Not extraction → **reconstruction with validation**
- Not heuristic → **formalized scoring + dependency graph**

Conclusion

This architecture redefines document understanding as a:

Hybrid Retrieval + Symbolic Reasoning + Computation Problem

It bridges the gap between:

- Information Retrieval
- Knowledge Representation
- Numerical Computation

3 System Overview

The proposed architecture is a **multi-stage hybrid pipeline** designed to transform unstructured financial documents into validated KPI datasets.

At a high level, the system decomposes into five sequential stages:

1. **Ingestion** — Multimodal extraction of document content
2. **Retrieval Preparation** — Metadata structuring + vectorization
3. **Hybrid Retrieval** — Symbolic + semantic search
4. **Structured Generation + Computation** — KPI reconstruction
5. **Validation + Monitoring** — Consistency + observability

Pipeline Abstraction:

$$\mathcal{D}_{pdf} \rightarrow \mathcal{X}_{multimodal} \rightarrow \mathcal{R}_{retrieval} \rightarrow \mathcal{G}_{structured} \rightarrow \mathcal{K}_{validated}$$

Interpretation:

- $\mathcal{X}_{multimodal}$: enriched document representation
- $\mathcal{R}_{retrieval}$: constraint-aware candidate selection
- $\mathcal{G}_{structured}$: schema-constrained generation
- $\mathcal{K}_{validated}$: mathematically consistent KPIs

Key Observation: Unlike standard pipelines, retrieval and computation are **tightly coupled**. The system does not retrieve answers — it retrieves **computable components**.

4 Multimodal Ingestion & Feature Extraction

This stage transforms raw PDF documents into a structured, multimodal representation suitable for downstream retrieval.

4.1 Adaptive Rasterization

To satisfy strict payload constraints (e.g., API limits), each page undergoes adaptive rasterization.

Constraint:

$$\text{ImageSize}(p, d) \leq S_{max}$$

where:

- p = page
- d = DPI
- S_{max} = size threshold (e.g., 5MB)

Optimization Process:

$$d_{t+1} = \alpha \cdot d_t \quad \text{with } \alpha \in (0, 1)$$

- Initial: $d_0 = 150$
- Iterative DPI reduction
- Horizontal splitting if constraint violated

Insight: This is a constrained optimization problem balancing:

- visual fidelity
- transmission feasibility

4.2 Vision LLM Extraction

Each rasterized page is processed by a Vision-Language Model:

$$\text{Page} \rightarrow (T, V, M)$$

where:

- T = textual transcription
- V = visual interpretation (graphs, tables)
- M = structured metadata

Important: The model performs both **perception** and **semantic structuring** in one step.

4.3 Page-Level Metadata Modeling

Each page is represented as a structured object:

$$M_{page} = \{E, P, K, I\}$$

with:

- E = detected entities (airlines)
- P = detected periods
- K = KPI candidates
- I = importance score

Importance Function:

$$I = f(\text{layout}, \text{KPI density}, \text{semantic relevance})$$

Role of Importance: Guides retrieval by prioritizing **high-value computational pages**.

4.4 Document-Level Consolidation

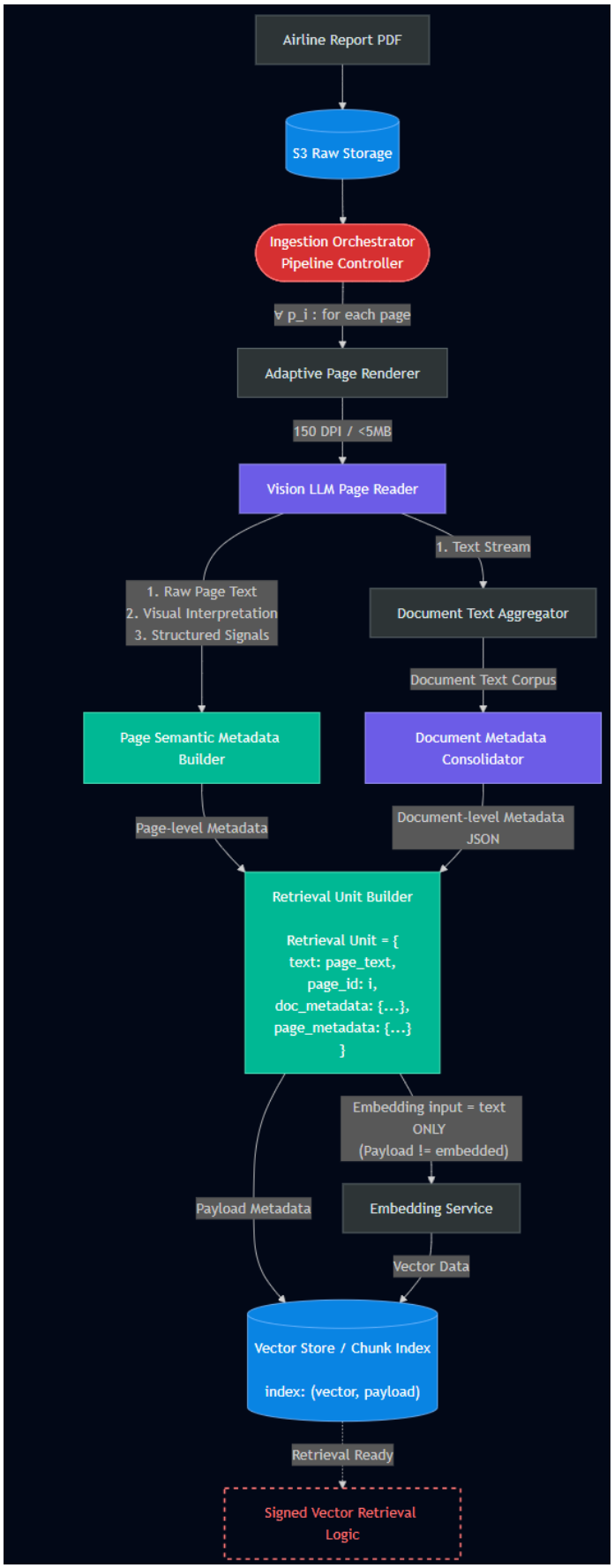
Page-level information is aggregated into a global document representation:

$$M_{doc} = \bigcup_{i=1}^{n} M_{page}^{(i)}$$

Two complementary mechanisms are used:

- **Regex extraction** → deterministic signals (entities, periods)
- **LLM arbitration** → resolves ambiguity and inconsistencies

Key Idea: Combine **deterministic parsing** with **probabilistic reasoning** to ensure both precision and flexibility.


Airline Report PDF
S3 Raw Storage
Ingestion Orchestrator
Pipeline Controller
∀ p_i : for each page
Adaptive Page Renderer
150 DPI / <5MB
Vision LLM Page Reader
1. Text Stream
1. Raw Page Text
2. Visual Interpretation
3. Structured Signals
Document Text Aggregator
Document Text Corpus
Page Semantic Metadata Builder
Document Metadata Consolidator
Page-level Metadata
Document-level Metadata JSON
Retrieval Unit Builder
Retrieval Unit = {
text: page_text,
page_id: i,
doc_metadata: {...},
page_metadata: {...}
}
Embedding input = text ONLY
(Payload != embedded)
Payload Metadata
Embedding Service
Vector Data
Vector Store / Chunk Index
index: (vector, payload)
Retrieval Ready
Signed Vector Retrieval Logic

Interpretation: This stage converts raw document content into:

- retrieval-ready chunks
- enriched metadata
- structured embedding inputs

5 Symbolic–Semantic Retrieval

This section introduces the core innovation of the system: a **hybrid retrieval paradigm** combining symbolic algebra and semantic similarity.

Core Idea: Retrieval is not based solely on similarity, but on **compositional completeness**.

5.1 Signed Vector Representation

Each KPI is represented in a symbolic space:

$$v \in \{-1, 0, 1, \text{None}\}^d$$

where each dimension corresponds to a KPI component.

Semantic Interpretation:

- $0 \rightarrow$ direct match (explicit KPI present)
- $\pm 1 \rightarrow$ complementary components
- None $\rightarrow$ irrelevant dimension

Insight: The vector encodes not only presence, but **computational role**.

5.2 Chunk Projection

Each document chunk is projected into the symbolic space:

$$M_{chunk} \rightarrow v_{chunk}$$

where M_{chunk} is the metadata extracted during ingestion.

This projection maps semantic signals into a structured algebraic representation.

5.3 Matching Logic

We define the retrieval condition as:

$$\text{Direct Hit} = (0) \ \vee \ (1 \wedge -1)$$

Interpretation:

- $0 \rightarrow$ KPI explicitly available
- $(1 \wedge -1) \rightarrow$ full reconstructibility

$$\text{Partial Hit} = \text{otherwise}$$

Key Property: Retrieval becomes a **logical inference problem**, not a nearest-neighbor search.

5.4 Complementary Retrieval

When only one side of a KPI is found:

$$1 \ \Rightarrow \ \text{search}(-1) \qquad -1 \ \Rightarrow \ \text{search}(1)$$

Mechanism:

- Identify missing component
- Trigger secondary retrieval
- Merge complementary chunks

Insight: This introduces **active retrieval**, guided by symbolic gaps.

5.5 Period Matching

Temporal alignment is enforced via classification:

$$p \in \{\text{exact}, \text{semi}, \text{partial}, \text{missed}\}$$

- **p-exact** $\rightarrow$ exact period match
- **p-semi-exact** $\rightarrow$ contains target + others
- **p-partial** $\rightarrow$ contributes to aggregation
- **p-missed** $\rightarrow$ irrelevant

Role: Ensures temporal consistency in KPI reconstruction.

5.6 Hybrid Scoring Function

Final ranking is computed as:

$$f = I + \lambda \cdot \cos(q, c) \quad \text{with } \lambda = 10$$

where:

- I = importance score (structural)
- $\cos(q, c)$ = semantic similarity

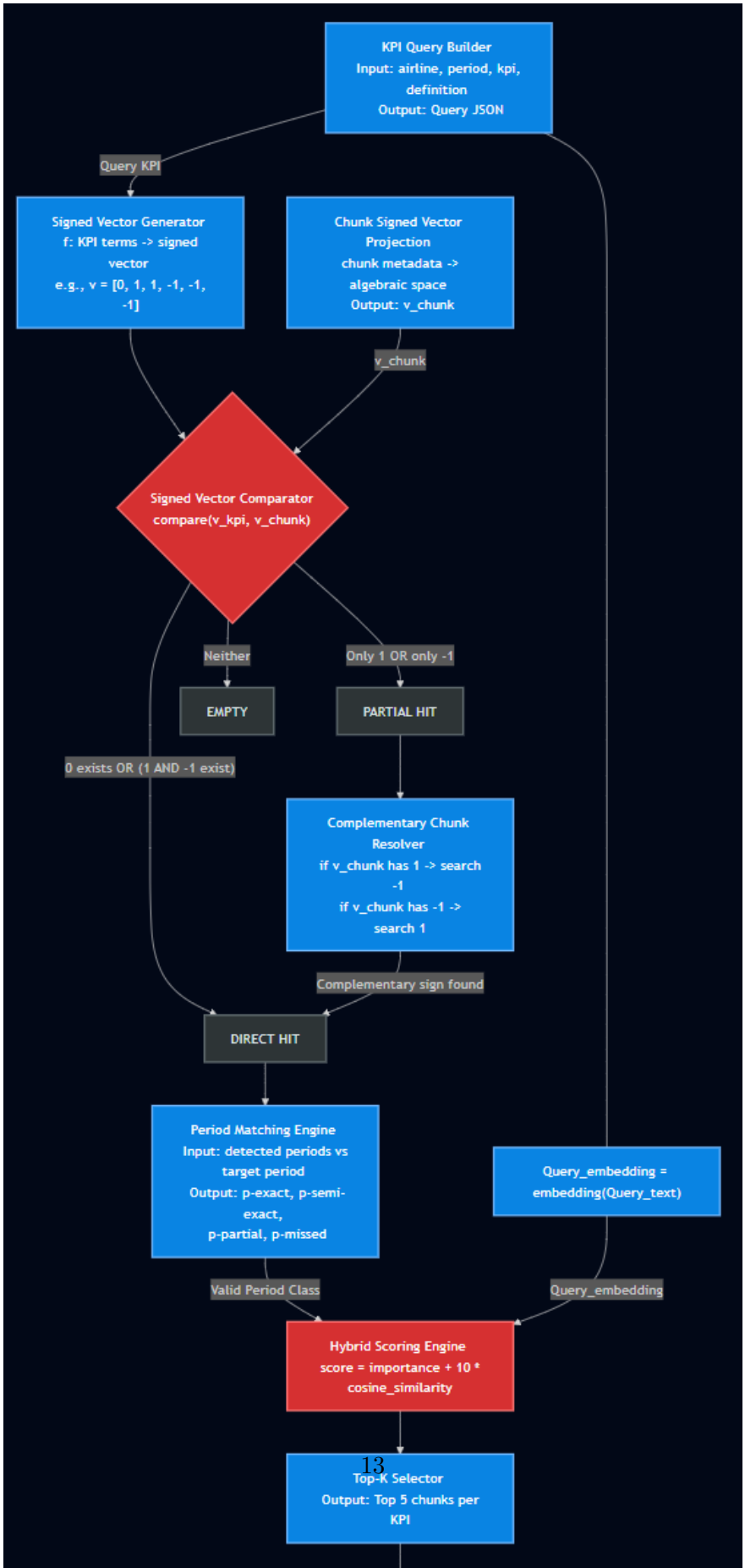
Interpretation:

- $I \rightarrow$ deterministic relevance
- $\cos(q, c) \rightarrow$ semantic alignment

Key Insight: The scoring function balances:

- **structure (what is needed)**
- **semantics (what looks similar)**

Signed Vector Retrieval Architecture



Interpretation: The system dynamically transitions between:

- symbolic reasoning (vector logic)
- semantic retrieval (embedding space)

6 KPI Extraction (Strict Mode)

This stage enforces **hallucination-free extraction**.

6.1 Retrieval Output

For each KPI query, the system retrieves:

$$\mathcal{C}_{top-k} = \{c_1, ..., c_k\}$$

where k is typically small (e.g., $k = 5$).

Goal: Provide the LLM with **high-signal, minimal-noise context**.

6.2 LLM Constraints

The extraction model operates under strict rules:

- **No computation allowed**
- **Only extraction from context**
- Classification:
  - explicit
  - computable
  - missing

$$\text{LLM}_{extract} : \mathcal{C}_{top-k} \rightarrow \mathcal{K}_{raw}$$

Critical Design: Decoupling extraction from computation eliminates hallucinations.

6.3 KPI Description Output

Each KPI is represented as:

$$K = (value, status, dependencies)$$

where:

- value → extracted text
- status → explicit / computable / missing
- dependencies → required variables

Insight: The output is not a value — it is a **computational state**.

7 Structured Generation

This stage converts unstructured textual outputs into a **strict, machine-valid schema**.

7.1 Schema Design

The transformation is defined as:

$$T \rightarrow \mathcal{J}_{strict}$$

where:

- T = extracted textual descriptions
- $\mathcal{J}_{strict}$ = schema-constrained JSON

Each KPI is encoded as:

$$K_i = \{id, name, value, status, timeframe, dependencies\}$$

Goal: Enforce **structural consistency** and eliminate ambiguous outputs.

7.2 Validation Loop

To ensure correctness, a validation loop is applied:

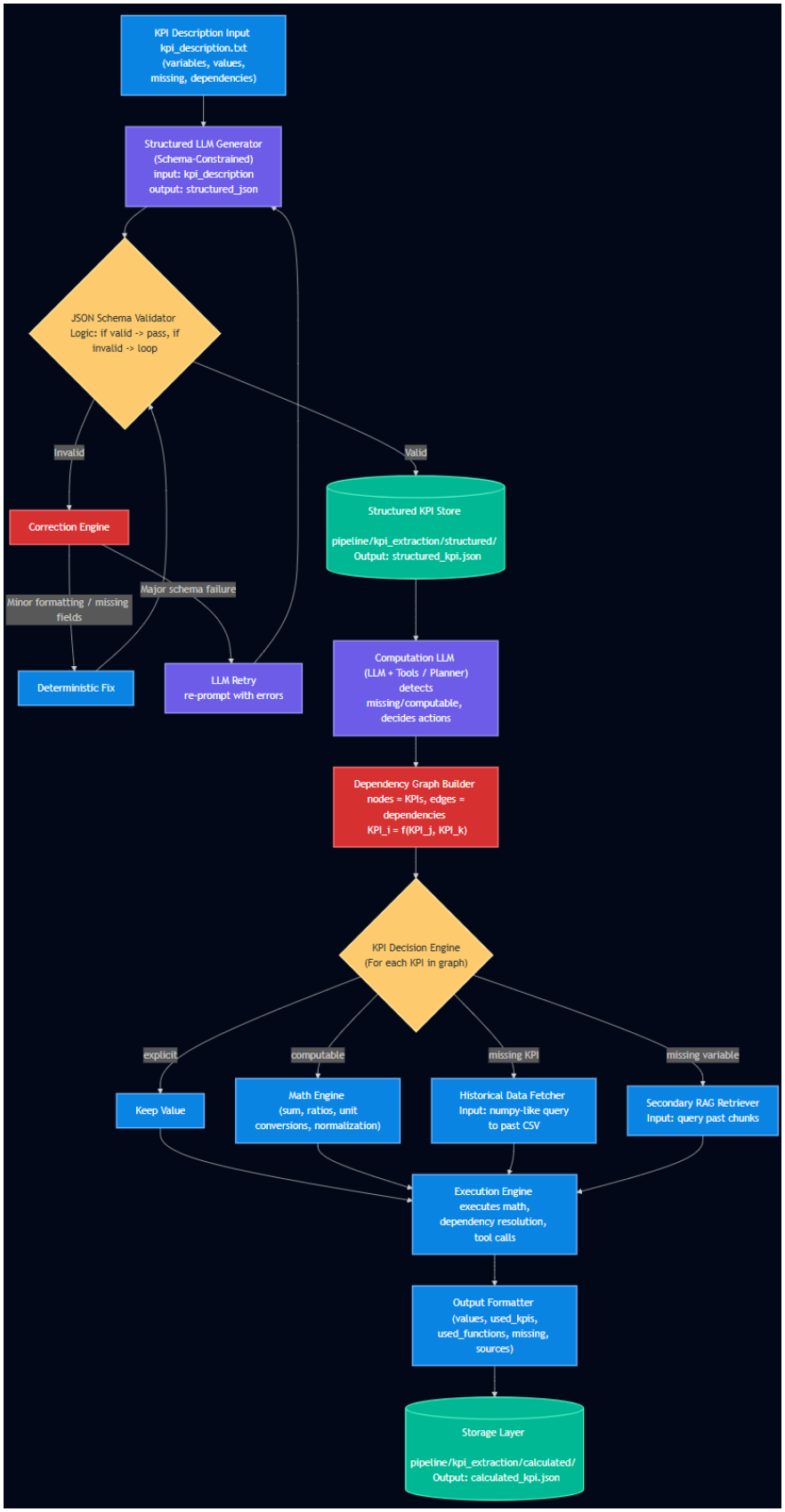
$$\text{invalid} \Rightarrow \text{retry}$$

Two correction mechanisms:

- **Deterministic Fix** → syntax, missing fields
- **LLM Retry** → semantic inconsistencies

Insight: Combining deterministic repair with probabilistic correction ensures both **precision** and **robustness**.

Structured Generation Pipeline



8 Computation Engine

This module transforms extracted variables into **fully computed KPIs**.

8.1 Problem Formulation

$$K_i = f(x_1, x_2, ..., x_n)$$

If K_i is missing:

$$\text{missing KPI} \Rightarrow \text{compute}(K_i)$$

Challenge: Required variables may be:

- partially available
- distributed across sources
- temporally misaligned

8.2 Dependency Graph

We define:

$$G = (V, E)$$

where:

- V = KPI nodes
- E = dependency edges

$$K_i \rightarrow \{x_j\}$$

Interpretation: KPI computation becomes a **graph traversal problem**.

8.3 Decision System

Each KPI is classified as:

$$\text{state}(K_i) \in \{\text{explicit}, \text{computable}, \text{missing}\}$$

- **explicit** $\rightarrow$ value available
- **computable** $\rightarrow$ dependencies satisfied
- **missing** $\rightarrow$ requires external retrieval

8.4 Tool System

The computation engine leverages specialized tools:

- **Math Engine** $\rightarrow$ arithmetic, normalization
- **Historical Fetch** $\rightarrow$ query past datasets
- **Secondary RAG** $\rightarrow$ retrieve missing variables

$$\mathcal{T} = \{\text{math}, \text{history}, \text{retrieval}\}$$

Key Idea: The LLM acts as a **planner**, not an executor.

8.5 Execution Engine

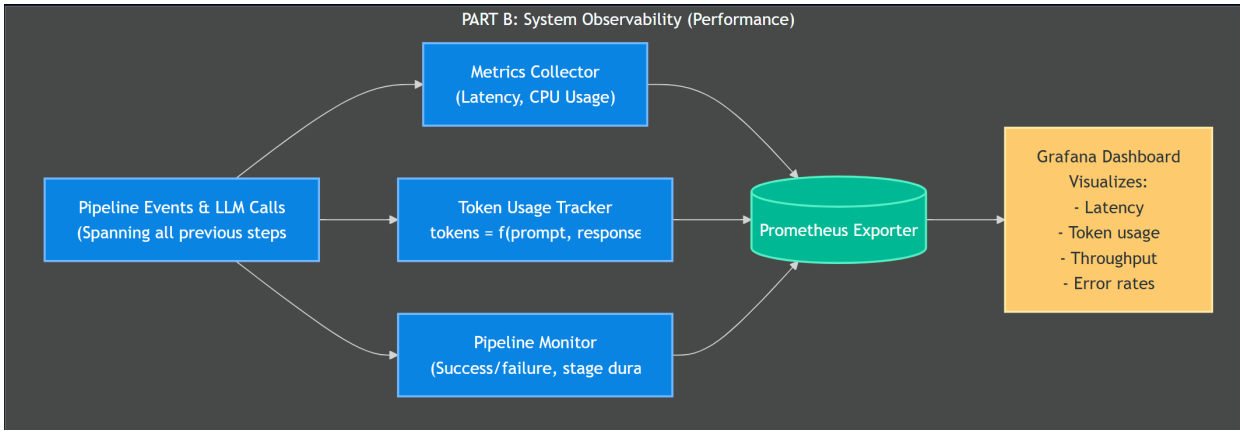
The system resolves the dependency graph:

$$G \rightarrow \text{topological execution}$$

$$\forall K_i \in V, \quad K_i = f(\text{resolved dependencies})$$

Insight: Computation follows a **deterministic execution order**, ensuring reproducibility.

Computation & Dependency Architecture



9 Validation & Observability

9.1 KPI Validation

Validation enforces mathematical consistency:

$$\text{constraints} \Rightarrow \text{verification}$$

Examples:

- sum constraints
- ratio consistency
- unit normalization

9.2 Accuracy Metric

$$\text{accuracy} = \frac{\text{valid KPIs}}{\text{total KPIs}}$$

Granularity: Metrics computed per:

- KPI
- airline
- period

9.3 Observability

The system tracks:

- Latency
- Token usage
- CPU utilization

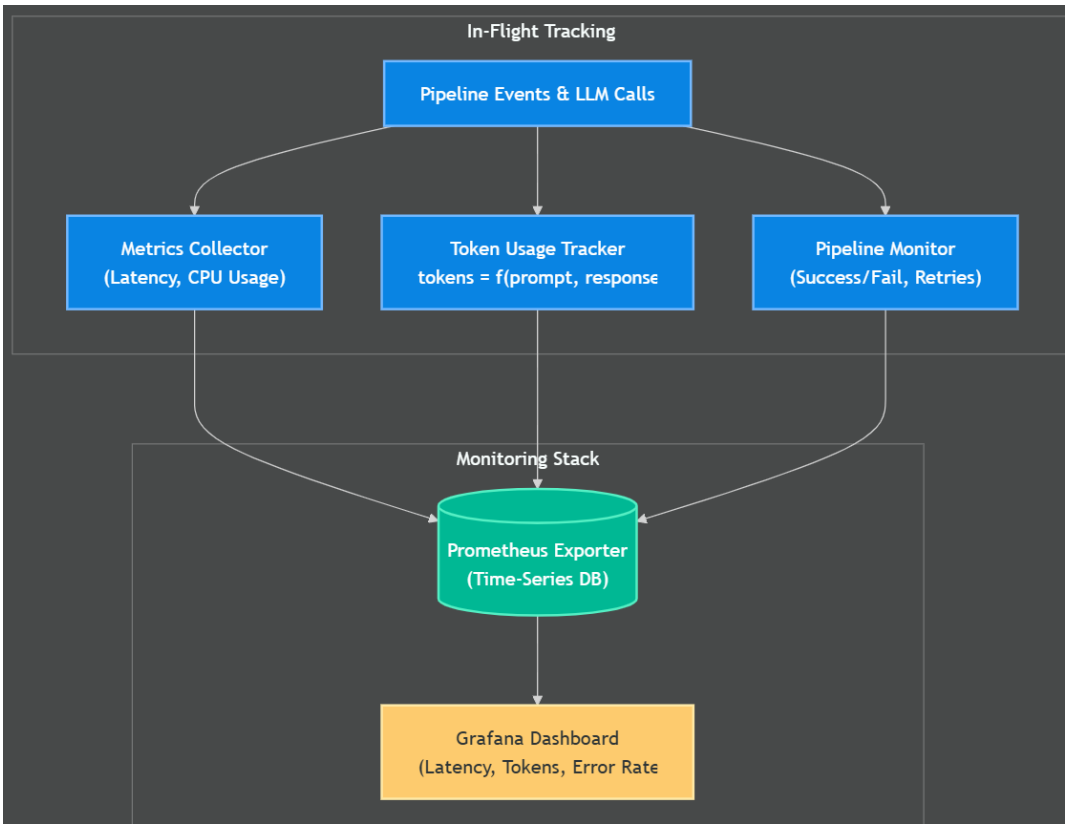
$$\mathcal{O} = \{\text{latency}, \text{tokens}, \text{compute}\}$$

9.4 Monitoring Stack

- Prometheus → metrics collection
- Grafana → visualization dashboards

Outcome: Full transparency on system performance and cost.

Validation & Monitoring Architecture



10 Results & Discussion

10.1 System Performance

- Processed PDFs: $\sim 10^2 - 10^3$
- Average latency: $\sim$ 7–20 minutes per document (instead of months mannualy)
- Overall accuracy: $\approx 95\% - 100\%$ (estimated)

10.2 Failure Cases

- LLM's Attention dillution in some rare cases

10.3 Limitations

- Sensitivity to very noisy pdfs
- Residual LLM uncertainty

10.4 Future Work

- Fine-tuned domain-specific models
- Enhanced symbolic reasoning layers
- Improved dependency graph optimization

Final Insight: This system demonstrates that document understanding evolves from:

$$\text{retrieval} \rightarrow \text{reasoning} \rightarrow \text{computation}$$

11 Productionization and Deployment

To ensure the system operates reliably beyond a research setting, a production layer was designed focusing on reproducibility, scalability, and robustness.

11.1 Containerization and CI/CD

The pipeline is fully containerized using Docker, enabling consistent execution across environments and simplifying deployment. Each component (ingestion, retrieval, computation) is encapsulated as an isolated service.

A CI/CD pipeline was implemented using Jenkins to automate the lifecycle:

$$\text{Code} \rightarrow \text{Build} \rightarrow \text{Test} \rightarrow \text{Deploy}$$

This ensures controlled updates, automated validation, and reproducible deployments through versioned container images.

11.2 Asynchronous Execution

Given the computational cost of multimodal processing and LLM inference, the pipeline follows an asynchronous execution model. Tasks are dispatched to background workers, allowing non-blocking processing and parallel handling of multiple documents.

Formally:

$$\text{Job} \rightarrow \{\text{Task}_1, ..., \text{Task}_n\}$$

This improves throughput and system responsiveness.

11.3 State Management and Persistence

The system maintains explicit state tracking across all stages. Data and intermediate artifacts are persisted in object storage (Amazon S3), including raw inputs, processed outputs, and audit logs.

Each job follows a lifecycle:

$$\text{Pending} \rightarrow \text{Processing} \rightarrow \text{Completed} \mid \text{Failed}$$

This enables traceability, failure recovery, and reproducibility of KPI reconstruction.

11.4 Deployment Architecture

The system is deployed in a cloud-native environment with a modular architecture:

$$\text{API Layer} \rightarrow \text{Workers} \rightarrow \text{Storage} + \text{Vector DB}$$

Key components include:

- Containerized compute services
- Amazon S3 for storage
- Vector database for retrieval
- Managed LLM APIs for inference

This design supports horizontal scaling and isolation of pipeline stages.

11.5 Production Readiness

The integration of containerization, CI/CD, asynchronous execution, and persistent state ensures:

- Scalability across multiple documents
- Robust failure handling and retries
- Reproducible execution
- Compatibility with monitoring systems (e.g., Prometheus, Grafana)

Outcome: The system transitions from a research prototype to a deployable, production-ready pipeline capable of handling real-world workloads.