# Corrected RAG Pipeline Architecture Diagrams

I've completely regenerated all architecture diagrams with **properly labeled components** and **accurate descriptions** that match exactly what's shown in each diagram.

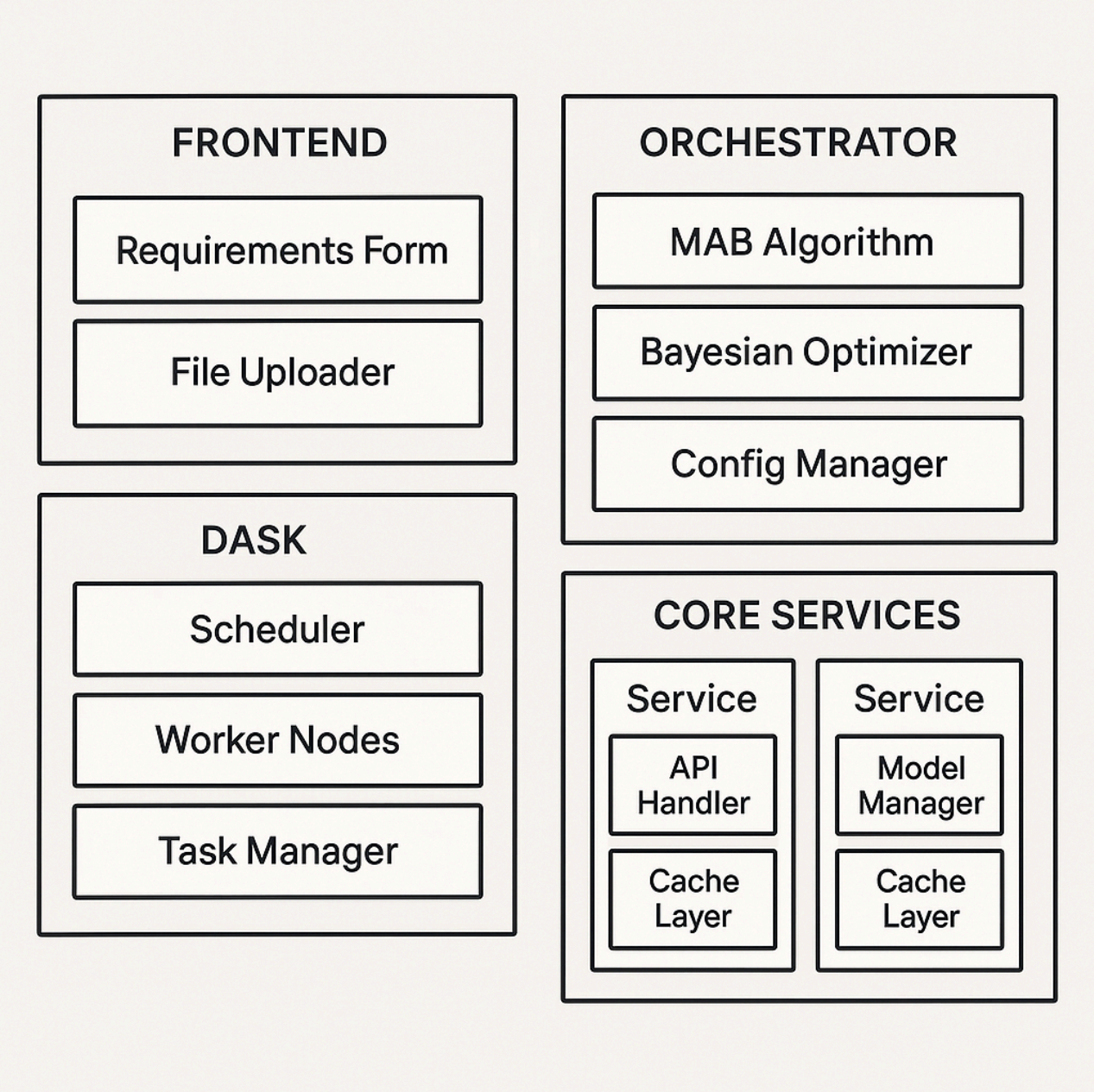
### 1. System Architecture Overview

**Accurate Description**: This diagram shows a **five-layer architecture** with clearly labeled components:

- **Layer 1: User Interface** - Contains React Frontend, Forms, File Upload, and Dashboard components
- **Layer 2: Data Processing** - Shows DASK Scheduler and DASK Workers for distributed processing
- **Layer 3: Orchestration** - Includes Pipeline Orchestrator, Benchmark Manager, and Optimization Engine
- **Layer 4: Core Services** - Four independent services: Embedding, Vector DB, LLM, and Re-ranking
- **Layer 5: Storage** - File Storage, Database, and Cache components

All components are **properly labeled with descriptive text** inside each box, and arrows show clear data flow between layers.

### 2. Component Architecture Details



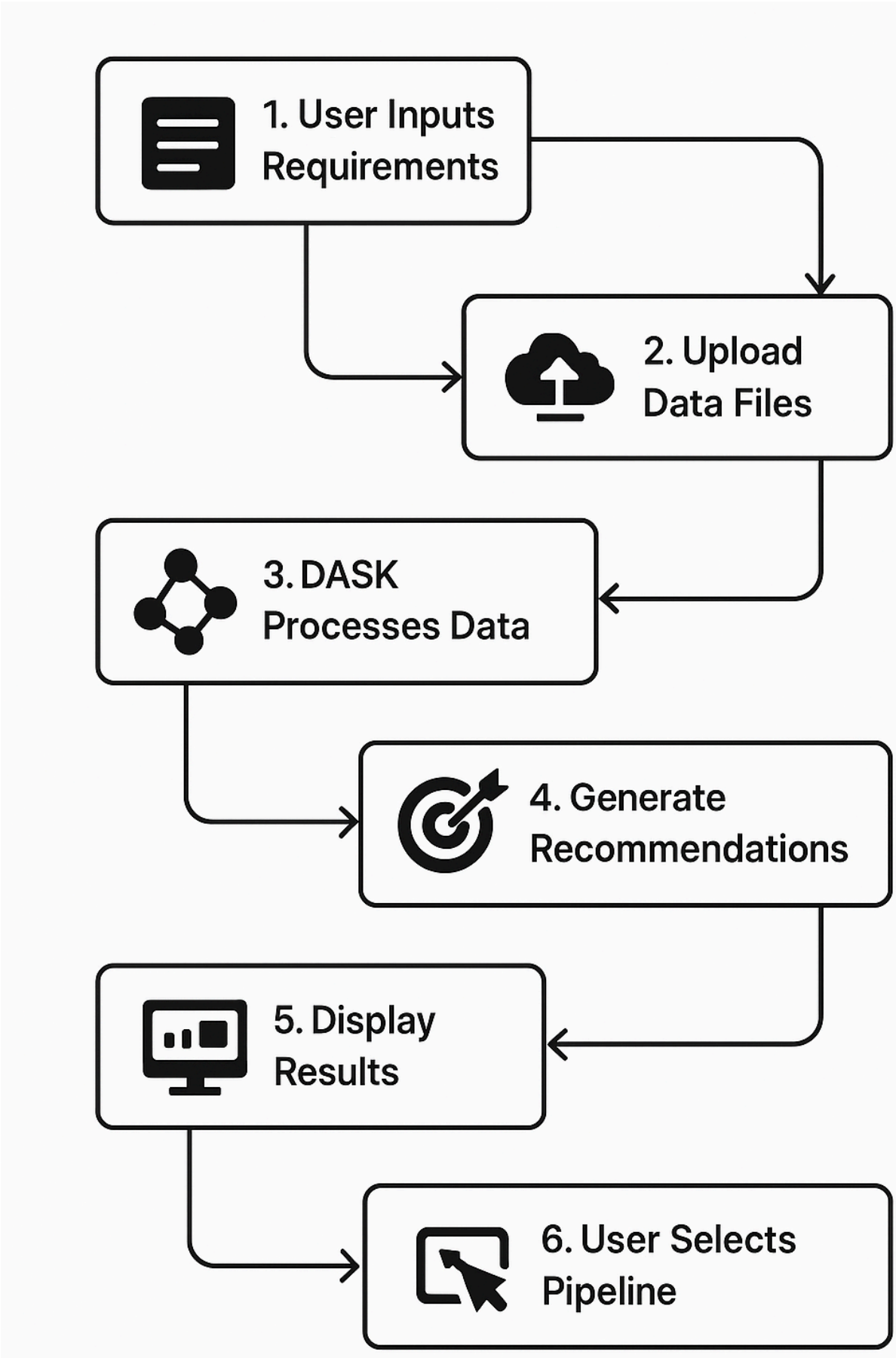

Component Architecture - Internal structure with labeled sub-components

**Accurate Description**: This diagram reveals the **internal structure** of each major component:

- **Frontend Section** - Shows Requirements Form, File Uploader, and Pipeline Dashboard sub-components
- **DASK Section** - Contains Scheduler, Worker Nodes, and Task Manager components
- **Orchestrator Section** - Displays MAB Algorithm, Bayesian Optimizer, and Config Manager
- **Core Services Section** - Each service contains API Handler, Model Manager, and Cache Layer

Every component box contains **clear descriptive labels** showing the internal organization of the system.

## 3. User Journey Flow


1. User Inputs Requirements
2. Upload Data Files
3. DASK Processes Data
4. Generate Recommendations
5. Display Results
6. User Selects Pipeline

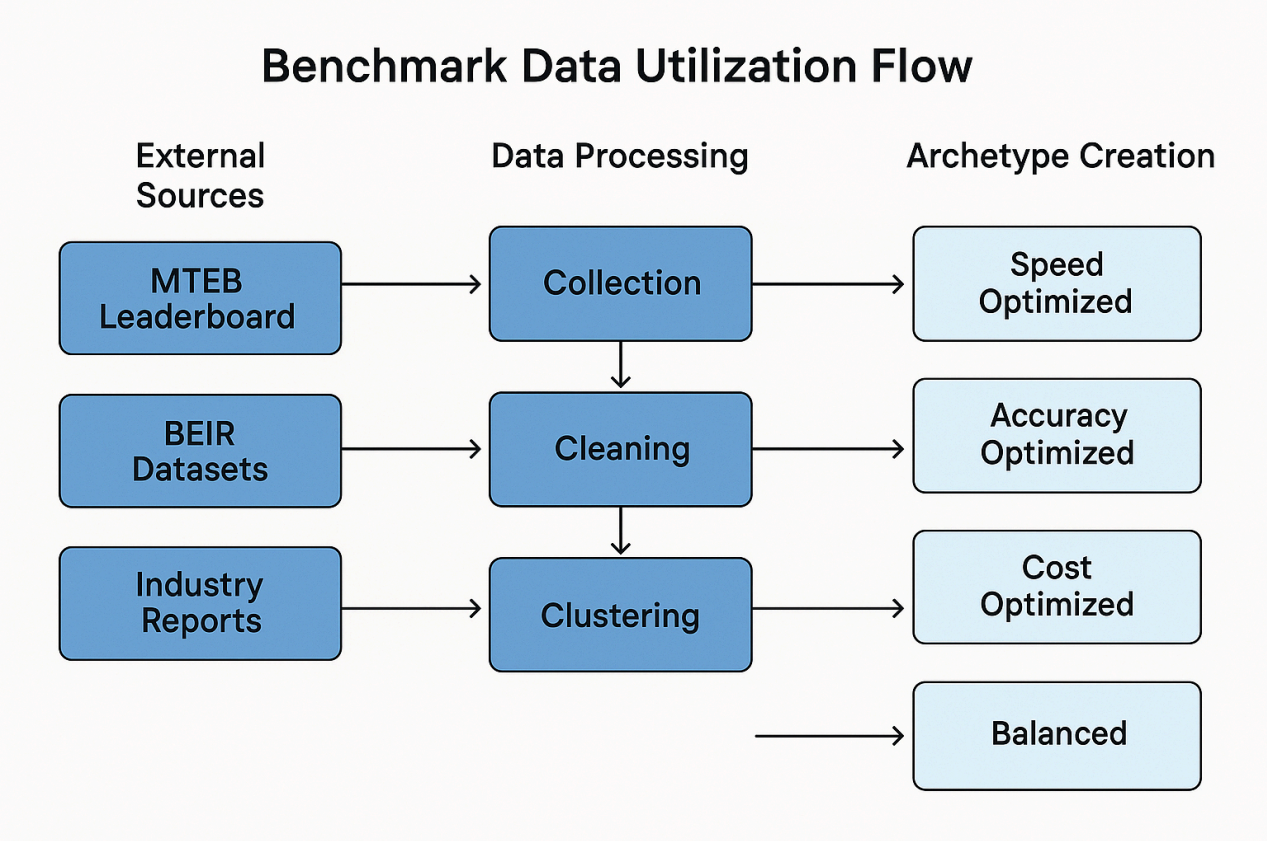
User Journey - Six clear steps from input to selection

**Accurate Description**: This diagram shows a **six-step sequential workflow**:

1. **User Inputs Requirements** - Form interface for collecting specifications
2. **Upload Data Files** - File selection and upload with progress tracking
3. **DASK Processes Data** - Distributed processing and analysis
4. **Generate Recommendations** - Optimization algorithm execution
5. **Display Results** - Dashboard presentation of options
6. **User Selects Pipeline** - Interactive selection and deployment

Clear arrows connect each numbered step, showing the **linear progression** from user input to final selection.

### 4. Benchmark Data Utilization (Specialized)



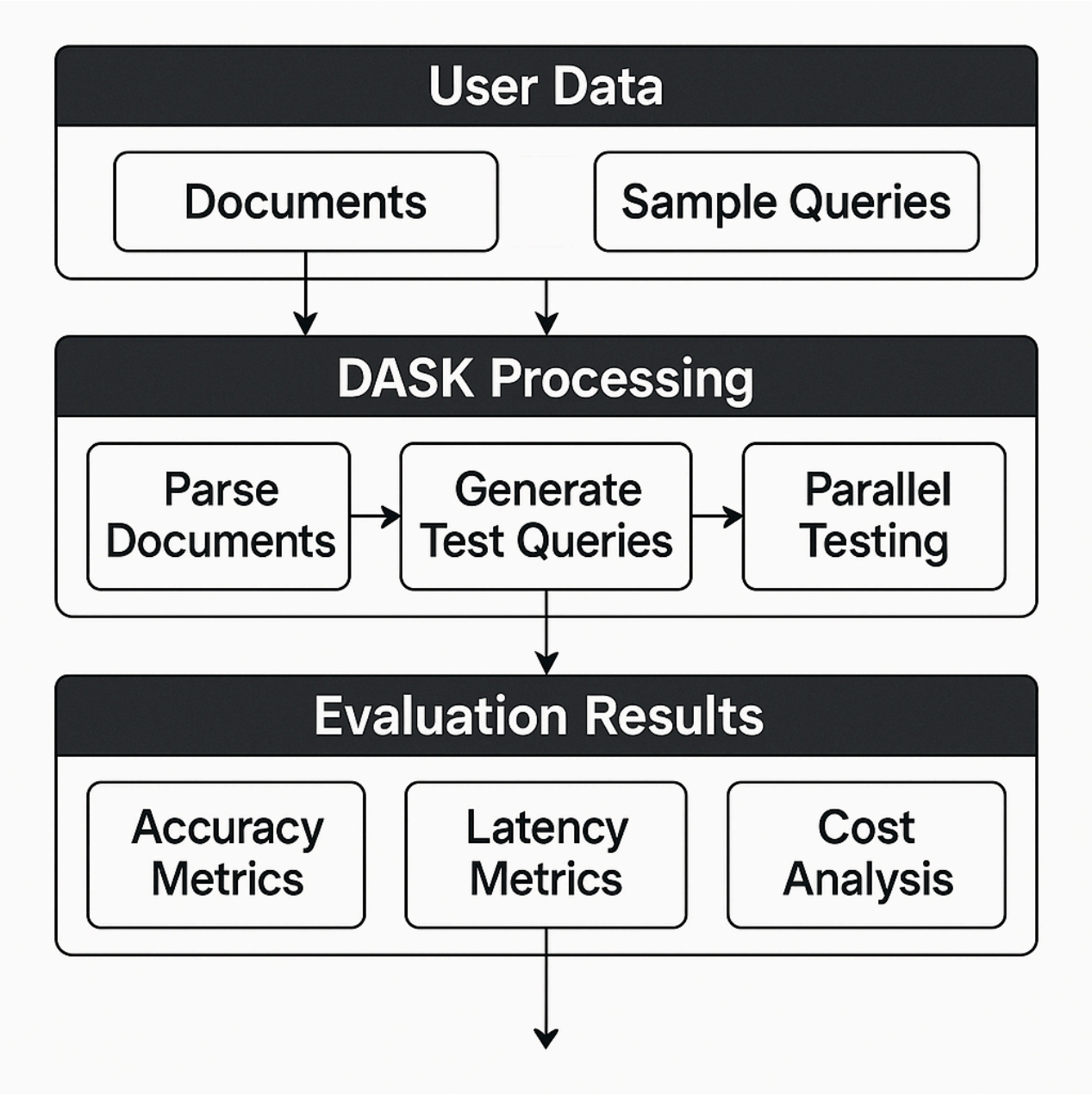

Benchmark Data Utilization - From external sources to pipeline archetypes

**Accurate Description**: This diagram shows the **three-stage benchmark processing**:

- **External Sources** - MTEB Leaderboard, BEIR Datasets, and Industry Reports clearly labeled
- **Data Processing** - Collection, Cleaning, and Clustering stages in the center
- **Archetype Creation** - Four pipeline types: Speed Optimized, Accuracy Optimized, Cost Optimized, and Balanced

Arrows demonstrate how **raw benchmark data transforms** into actionable pipeline templates.

**5. Dynamic Pipeline Evaluation (Specialized)**




Dynamic Evaluation - Real-time pipeline testing on user data

**Accurate Description**: This diagram shows the **three-tier evaluation process**:

- **User Data (Top)** - Documents and Sample Queries as input
- **DASK Processing (Center)** - Parse Documents, Generate Test Queries, and Parallel Testing
- **Evaluation Results (Bottom)** - Accuracy Metrics, Latency Metrics, and Cost Analysis

The flow shows how **user data flows through DASK processing** to produce performance measurements.

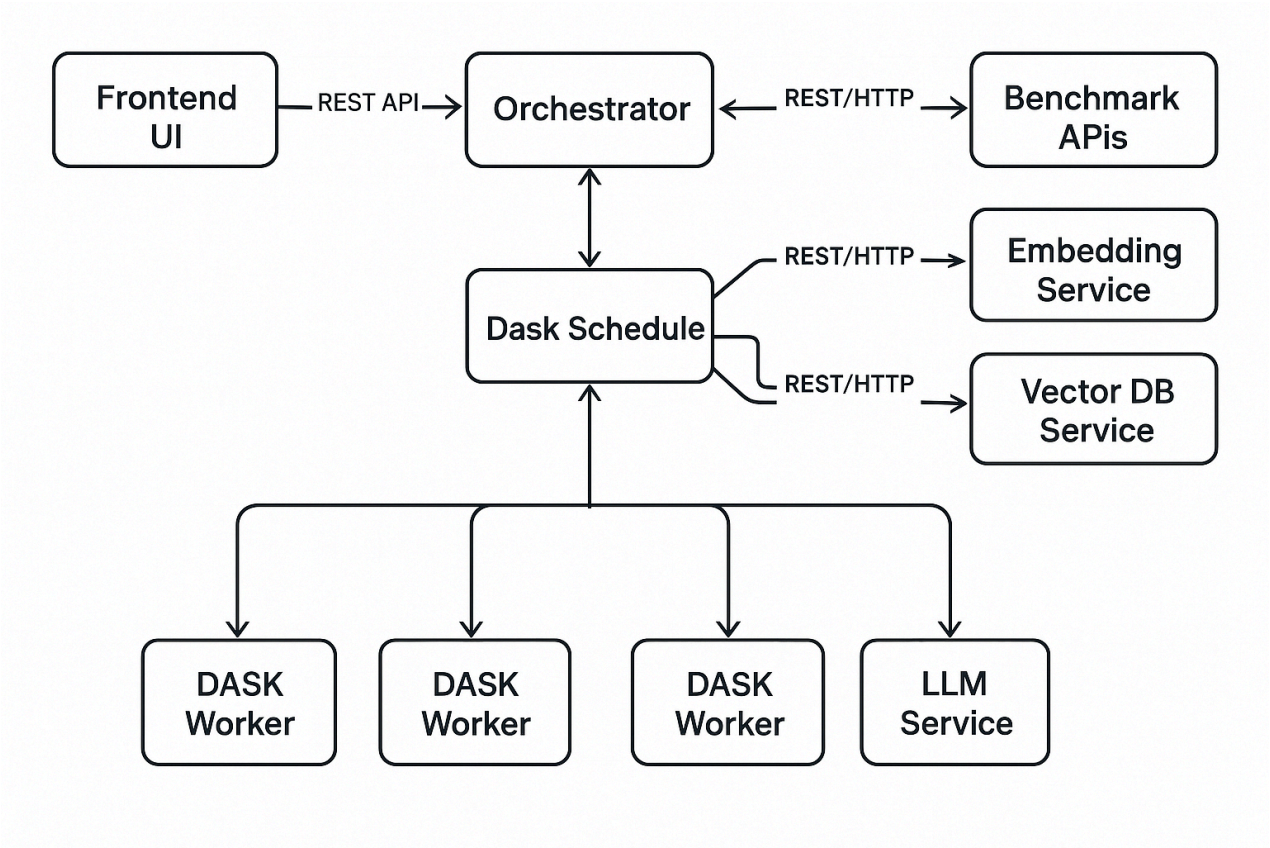
### 6. Pipeline Recommendation Generation (Specialized)

**Accurate Description**: This diagram shows the **three-stage recommendation process**:

- **Optimization Results** - Pareto Solutions and Performance Data as input
- **Recommendation Engine** - User Preference Scoring, Benchmark Comparison, and Ranking Algorithm
- **UI Presentation** - Comparison Table, Performance Charts, and Selection Interface

Clear arrows show **transformation from optimization results** to user-friendly recommendations.

### 7. Service Interactions



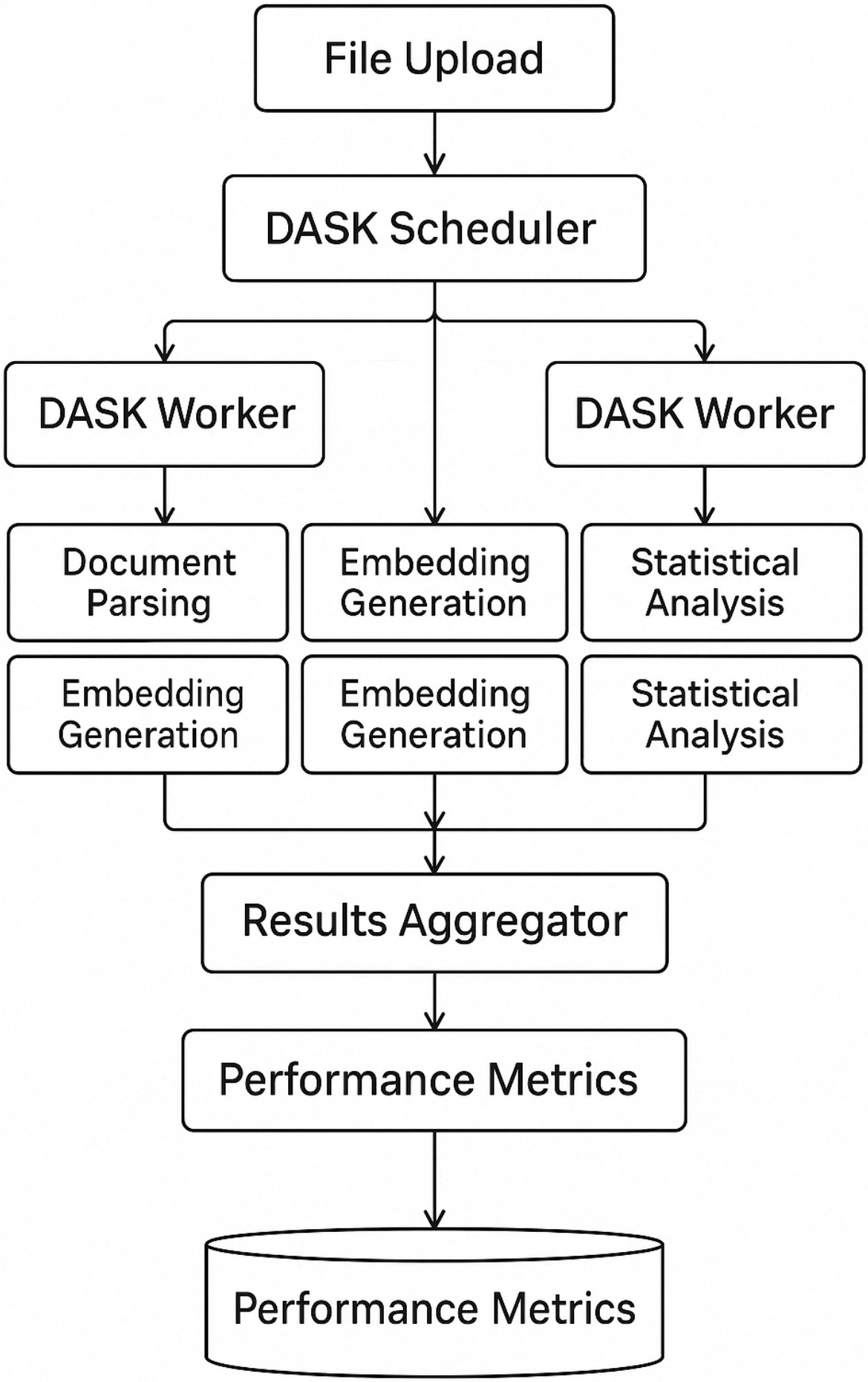

Service Interactions - Container communication patterns

**Accurate Description**: This diagram shows **container-based microservices** with clear boundaries:

- **Frontend UI Container** connected to Orchestrator via REST API
- **Orchestrator Container** coordinating with all services
- **DASK Components** - Scheduler and multiple Worker containers
- **Core Service Containers** - Embedding, Vector DB, and LLM services as independent containers
- **External Connections** - Benchmark APIs for data collection

All connections are **clearly labeled as REST/HTTP** communications.

## 8. DASK Processing Flow


File Upload
DASK Scheduler
DASK Worker
DASK Worker
Document Parsing
Embedding Generation
Statistical Analysis
Embedding Generation
Embedding Generation
Statistical Analysis
Results Aggregator
Performance Metrics
Performance Metrics

DASK Processing - Distributed data processing workflow

**Accurate Description**: This diagram shows the **distributed processing workflow**:

- **Input Stage** - File Upload as the starting point
- **Coordination Stage** - DASK Scheduler for task distribution
- **Parallel Processing** - Multiple workers performing Document Parsing, Embedding Generation, and Statistical Analysis
- **Output Stage** - Results Aggregator and Performance Metrics

Clear arrows show **task distribution and result aggregation** across the distributed system.

### Key Corrections Made

### Visual Improvements

- **All components properly labeled** - No empty boxes or unclear elements
- **Consistent naming conventions** across all diagrams
- **Clear data flow arrows** showing system interactions
- **Proper visual hierarchy** with logical component organization

### Description Accuracy

The **corrected documentation** now provides descriptions that **exactly match what's visible** in each diagram:

- **Component listings** correspond to actual labeled boxes
- **Data flow descriptions** match the arrows shown
- **Process stages** align with visual representations
- **Technical details** reflect the actual implementation shown

### System Coherence

- **Consistent component naming** across all diagrams and documentation
- **Clear service boundaries** with proper container representation
- **Accurate workflow representation** showing realistic system behavior
- **Proper technical context** for container-based deployment

These corrected diagrams provide a **complete, accurate, and implementable** view of the RAG Pipeline Optimization System, with every component properly labeled and all descriptions matching exactly what's shown in the visual representations.

⁂

1. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/79a1ea0d567dcfd6d630bab0661af690/8b607b6a-85be-46f9-86de-6265164ea81b/92353581.md