Corrected RAG Pipeline Architecture Diagrams - Accurate Descriptions

Overview

This document provides accurate descriptions that match the corrected architecture diagrams for the RAG Pipeline Optimization System. All diagrams have been regenerated with properly labeled components and no empty boxes.

1. System Architecture Overview

Diagram: [117]

What the diagram shows: A five-layer architecture with clearly labeled components in each layer.

Layer 1: User Interface

- React Frontend: Web application interface
- Forms: User requirement input forms
- File Upload: Document upload interface
- Dashboard: Pipeline comparison and selection interface

Layer 2: Data Processing

- DASK Scheduler: Task coordination service
- DASK Workers: Distributed processing nodes

Layer 3: Orchestration

- Pipeline Orchestrator: Central coordination service
- Benchmark Manager: External data integration
- Optimization Engine: Algorithm execution

Layer 4: Core Services

- Embedding Service: Text-to-vector conversion
- Vector DB Service: Similarity search
- LLM Service: Text generation
- Re-ranking Service: Result refinement

Layer 5: Storage

- File Storage: Document storage

- Database: Configuration and metadata
- Cache: Performance optimization

Data Flow: Arrows show upward flow from storage through services to user interface, with bidirectional communication between orchestration and other layers.

2. Component Architecture Details

Diagram: [118]

What the diagram shows: Internal structure of each major system component with sub-components clearly labeled.

Frontend Section

- Requirements Form: User input collection
- File Uploader: Document upload handling
- Pipeline Dashboard: Results presentation

DASK Section

- Scheduler: Task distribution
- Worker Nodes: Processing execution
- Task Manager: Job coordination

Orchestrator Section

- MAB Algorithm: Multi-armed bandit optimization
- Bayesian Optimizer: Parameter tuning
- Config Manager: Configuration handling

Core Services Section
Each service (Embedding, Vector DB, LLM, Re-ranking) contains:

- API Handler: Request processing
- Model Manager: AI model management
- Cache Layer: Performance caching

Integration: Clear connections between components showing data and control flow.

3. User Journey Flow

Diagram: [119]

What the diagram shows: A six-step sequential workflow with numbered steps and descriptive labels.

Step 1: User Inputs Requirements

- Form interface for collecting use case details
- Priority settings for accuracy, speed, cost
- Constraint specifications

Step 2: Upload Data Files

- File selection and upload interface
- Support for multiple document formats
- Progress tracking during upload

Step 3: DASK Processes Data

- Distributed processing of uploaded files
- Document parsing and analysis
- Statistical computation

Step 4: Generate Recommendations

- Optimization algorithm execution
- Pipeline configuration generation
- Performance prediction

Step 5: Display Results

- Recommendation presentation in dashboard
- Performance comparisons
- Cost analysis

Step 6: User Selects Pipeline

- Interactive selection interface
- Configuration customization options
- Deployment preparation

Flow: Clear arrows connect each step sequentially, showing the linear progression from user input to final selection.

4. Benchmark Data Utilization

Diagram: [120]

What the diagram shows: A three-stage process converting external benchmark data into pipeline archetypes.

External Sources (Left Side)

- MTEB Leaderboard: Academic embedding benchmarks
- BEIR Datasets: Retrieval task benchmarks

- Industry Reports: Commercial performance data

Data Processing (Center)

- Collection: Automated data gathering
- Cleaning: Data normalization and validation
- Clustering: Grouping similar configurations

Archetype Creation (Right Side)

- Speed Optimized: Low-latency configurations
- Accuracy Optimized: High-precision configurations
- Cost Optimized: Budget-friendly configurations
- Balanced: Multi-objective optimized configurations

Data Flow: Arrows show progression from external sources through processing to final archetype creation, demonstrating how raw benchmark data becomes actionable pipeline templates.

5. Dynamic Pipeline Evaluation

Diagram: [121]

What the diagram shows: A three-tier evaluation process using user data to test pipeline performance.

User Data (Top Tier)

- Documents: Uploaded user files
- Sample Queries: Generated test questions

DASK Processing (Center Tier)

- Parse Documents: Text extraction and cleaning
- Generate Test Queries: Automatic query creation
- Parallel Testing: Simultaneous pipeline evaluation

Evaluation Results (Bottom Tier)

- Accuracy Metrics: Performance measurements
- Latency Metrics: Response time analysis
- Cost Analysis: Expense calculations

Processing Flow: Arrows show data flowing from user inputs through DASK processing to evaluation outputs, demonstrating real-time performance assessment.

6. Pipeline Recommendation Generation

Diagram: [122]

What the diagram shows: A three-stage process transforming optimization results into user-friendly recommendations.

Optimization Results (Left Side)

- Pareto Solutions: Multi-objective optimal configurations
- Performance Data: Predicted metrics for each solution

Recommendation Engine (Center)

- User Preference Scoring: Alignment with user priorities
- Benchmark Comparison: Industry positioning
- Ranking Algorithm: Solution ordering

UI Presentation (Right Side)

- Comparison Table: Side-by-side performance display
- Performance Charts: Visual performance representation
- Selection Interface: Interactive choice mechanism

Processing Flow: Arrows show transformation from raw optimization results through intelligent processing to polished user presentation.

7. Service Interactions

Diagram: [123]

What the diagram shows: Container-based microservices communication patterns with clear service boundaries.

Frontend UI Container

- Web interface serving user interactions
- Connected to orchestrator via REST API

Orchestrator Container

- Central coordination service
- REST API connections to all other services

DASK Components

- DASK Scheduler container for task coordination
- Multiple DASK Worker containers for parallel processing

Core Service Containers

- Embedding Service: Independent container
- Vector DB Service: Independent container
- LLM Service: Independent container

External Connections

- Benchmark APIs: External data sources

Communication: All connections clearly labeled as REST/HTTP, showing stateless service interactions.

8. DASK Processing Flow

Diagram: [124]

What the diagram shows: Distributed data processing workflow with parallel execution across multiple workers.

Input Stage

- File Upload: Initial data ingestion point

Coordination Stage

- DASK Scheduler: Central task distribution

Parallel Processing Stage

- Multiple DASK Workers executing tasks simultaneously:
  - Document Parsing: Text extraction
  - Embedding Generation: Vector creation
  - Statistical Analysis: Data profiling

Output Stage

- Results Aggregator: Combining worker outputs
- Performance Metrics: Final processing statistics

Workflow: Clear arrows show task distribution from scheduler to workers and result aggregation back to central output.

Technical Accuracy Notes

Component Labeling

- All diagram components contain descriptive text labels
- No empty boxes or unlabeled elements
- Consistent naming conventions across all diagrams

Flow Representation

- Arrows clearly indicate data and control flow direction
- Sequential processes numbered where appropriate
- Parallel processes visually distinguished

Service Boundaries

- Clear separation between different system components
- Container boundaries explicitly shown
- API connections clearly marked

Data Flow Accuracy

- Input and output stages clearly identified
- Processing transformations explicitly shown
- Result aggregation points clearly marked

Deployment Context

These diagrams represent a container-based deployment model suitable for:

- Docker Compose deployment on single or multiple servers
- Cloud-based container orchestration
- Development and testing environments
- Production deployment without Kubernetes complexity

All diagrams accurately reflect the technical implementation described in the accompanying architecture documentation, with consistent component naming and clear visual representation of system interactions.