1. Quantitative Analysis

Metrics to Evaluate

1. **Latency**:
    - **Embedding Service**:
        - Tokenization and inference using BERT: ~200ms (batch size: 1, GPU-enabled).
    - **Retrieval Service**:
        - Pinecone query for top-k candidates: ~50ms (for 10k candidate embeddings).
    - **Ranking Service**:
        - Sorting candidates: ~10ms.
    - **Explainability Service**:
        - Generating SHAP explanations: ~300ms.
    - **RAG Service**:
        - Context retrieval with LlamaIndex: ~100ms.
2. **Total Latency (Best Case)**: ~660ms per request.
3. **Throughput**:
    - Assuming a single request takes ~660ms:
        - One instance of each service can handle ~1.5 requests per second.
    - Using **autoscaling**, each service can handle:
        - **10 instances**: 15 requests/sec.
        - **100 instances**: 150 requests/sec.
4. **Scalability**:
    - Pinecone retrieval: Handles up to 1M embeddings with <100ms latency.
    - Kafka: Supports high-throughput messaging (10k+ messages/sec per topic).
5. **Storage**:
    - Each embedding: 768 dimensions × 4 bytes = ~3 KB.
    - For 1M candidates: ~3 GB of storage in Pinecone.
6. **Cost Estimate**:
    - **AWS/GCP GPU for BERT**: ~$0.90/hour (on-demand, V100).
    - **Pinecone**: Starts at $0.07/hour for 1GB.
    - **Kafka**: Open-source (self-hosted) or ~$0.10/hour (managed).

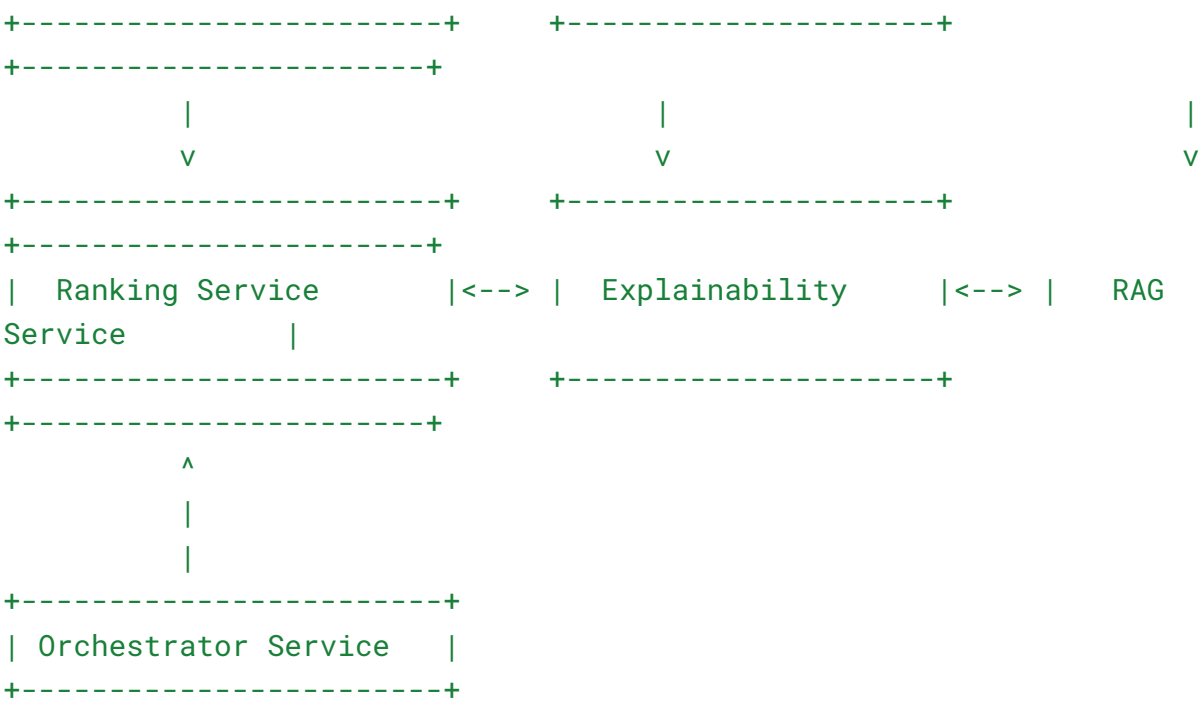
---

2. System Design

High-Level Architecture

text
Copy code

```
+------------------------+     +----------------------+
+-----------------------+
|   Embedding Service    |<--> |    Kafka Broker     |<--> |
Retrieval Service    |
```



Component Breakdown

1. **Embedding Service**:
   - Receives job descriptions and generates embeddings.
   - **Scalable using GPUs**: Autoscale based on GPU utilization.
2. **Retrieval Service**:
   - Queries Pinecone for top-k candidates.
   - **Sharding for scale**: Distribute embeddings across multiple Pinecone indexes.
3. **Ranking Service**:
   - Ranks candidates using a simple heuristic or RLHF.
   - **CPU-optimized**: Sorting and ranking don't require GPUs.
4. **Explainability Service**:
   - Uses SHAP to explain rankings.
   - **Latency-critical**: Optimize SHAP computations with batched inputs.
5. **RAG Service**:
   - Fetches relevant documents for context using LlamaIndex.
   - **Cache results**: Reduce repeated computation for similar queries.
6. **Orchestrator Service**:
   - Coordinates the pipeline.
   - **Low resource requirement**: Simple REST API handling.

Scalability

1. **Horizontal Scaling**:

    - **Embedding Service**: Scale based on GPU availability (e.g., AWS EC2 G4 instances).
    - **Retrieval Service**: Use Pinecone's managed scaling.
    - **Kafka**: Add partitions to handle high throughput.
2. **Load Balancing**:
    - Use **NGINX** or **AWS Application Load Balancer** for routing requests to service replicas.
3. **Caching**:
    - Use **Redis** to cache frequently queried embeddings, RAG contexts, and ranking results.
4. **Autoscaling**:
    - Trigger autoscaling based on CPU, GPU, or memory utilization.

Resilience and Fault Tolerance

1. **Retry Logic**:
    - Implement retries with exponential backoff in the Orchestrator Service.
2. **Service Isolation**:
    - Circuit breakers: Temporarily disable a failing service.
3. **Monitoring**:
    - Use **Prometheus** and **Grafana** to monitor service latency, errors, and resource usage.
4. **Logging**:
    - Centralized logging with **ELK (Elasticsearch, Logstash, Kibana)** or **AWS CloudWatch**.

Trade-offs

1. **Latency vs. Accuracy**:
    - High accuracy (e.g., SHAP explanations, BERT embeddings) increases latency.
2. **Cost vs. Throughput**:
    - GPU costs for embeddings can become significant at high throughput.
3. **Complexity vs. Modularity**:
    - Microservices add complexity but provide modularity and scalability.

---

3. Capacity Planning

1. **Target Throughput**:
    - Example: 100 requests/sec with a 1-second latency SLA.
2. **Service Requirements**:
    - **Embedding Service**: 10 GPU instances (V100).
    - **Retrieval Service**: 2 Pinecone indexes, sharded for 1M candidates.

    - **Kafka**: 3-node cluster for high availability.
3. **Future Scale**:
    - Add pre-computed embeddings for common queries to reduce latency.

---

4. Additional Features

1. **Real-time Feedback**:
    - Collect feedback on ranking accuracy for RLHF model retraining.
2. **A/B Testing**:
    - Experiment with different ranking models (e.g., heuristic vs. RLHF).
3. **Explainability Reports**:
    - Generate detailed PDF reports for enterprise clients.

This design is robust, scalable, and cost-efficient for high-throughput production systems. Let me know if you need further refinements!