Smart Agentic Task Router with Neo4j-Enhanced Agent Profiling (Updated Proposal)

Incorporating Instructor Feedback & Filling All Gaps

Student: Azad Jagtap **Domain:** Multi-Agent Systems & Knowledge Representation

---

1. Problem Statement

Modern AI workflows increasingly depend on **specialized agents**—such as web searchers, code analyzers, summarizers, and data visualizers—each optimized for distinct tasks. Users, however, often struggle to determine which agent best fits their query. This leads to: - Suboptimal results - Inefficient resource usage - Poor user experience

Existing routing approaches (keyword matching, intent classification) fail to model the **nuanced relationships** between: - Task requirements - Agent capabilities - Performance histories - Complexity levels - Agent interoperability

Core Problem:

> *How do we intelligently route user queries to the most appropriate specialized agent using a knowledge graph that supports explainability, adaptability, and continuous learning?*

---

2. Proposed Solution: Smart Agentic Router

The Smart Agentic Router uses **Neo4j**, **RDF/OWL modeling**, **SHACL constraints**, and a **multi-agent architecture** to: 1. Model agent capabilities, relationships, fallback logic 2. Understand task types and query semantics 3. Execute SPARQL/Cypher queries to rank and select the optimal agent 4. Learn over time using performance feedback and routing outcomes

The system provides **explainable routing** by exposing the graph traversal path used in each decision.

---

3. Knowledge Graph Design (Expanded with Instructor Feedback)

3.1 Core Classes (RDF/OWL)

Agent Classes

- **ex:Agent**: Generic agent

- **ex:SpecializedAgent** ⟵ ex:Agent
- **ex:RouterAgent** ⟵ ex:Agent

Task & Query Classes

- ex:Task
- ex:TaskType
- ex:Query

Capability & Performance Classes

- ex:Capability
- ex:RoutingDecision
- ex:PerformanceRecord

3.2 Key Relationships

Agent–Capability

- ex:hasCapability
- ex:capabilityLevel (0.0–1.0)
- ex:fallbackAgent
- ex:similarTo

Task–Capability

- ex:requiresCapability
- ex:complexityLevel

Query–Agent

- ex:routedTo
- ex:confidence

Performance

- ex:successfullyHandled
- ex:failureCount
- ex:latency

3.3 Additional RDF Properties (Added per Feedback)

- **ex:inputFormat**
- **ex:outputFormat**
- **ex:domainExpertise**
- **ex:historicalAccuracy**

4. Expanded SHACL Shapes

Based on instructor feedback, new shapes are included.

4.1 Agent Shape (Revised)

- Must have name
- Must have ≥1 capability
- capabilityLevel ∈ [0.0, 1.0]
- Each SpecializedAgent must have ≥1 fallback (SPARQL constraint)

4.2 Task Shape (New)

- Must have required capability
- Must have complexity level in [0.0, 1.0]
- Must map to at least one TaskType

4.3 Capability Shape (New)

- Must define capability type
- Must be referenced by at least one agent

4.4 Query Shape (New)

- Must have entity-extracted attributes
- Must route to exactly one agent

4.5 SPARQL Business Rules

Example:

```
# Every SpecializedAgent must have at least one fallback
ASK WHERE {
  ?agent a ex:SpecializedAgent .
  FILTER NOT EXISTS { ?agent ex:fallbackAgent ?fb . }
}
```

5. Required Cypher Queries (Updated for Neo4j)

Instructor requested operational queries; these are now rewritten in **Cypher**, optimized for Neo4j.

Query 1: Select Best Agent by Capability Level

```
MATCH (agent:Agent)-[:HAS_CAPABILITY]->(cap:Capability),
      (task:Task)-[:REQUIRES_CAPABILITY]->(cap)
WHERE agent.capabilityLevel > 0.7
RETURN agent, agent.capabilityLevel AS capLevel
ORDER BY capLevel DESC;
```

Query 2: Get Similar Agents for Fallback

```
MATCH (agent:Agent)-[:SIMILAR_TO]->(fallback:Agent)
RETURN fallback;
```

Query 3: Retrieve Historical Routing Performance

```
MATCH (:RoutingDecision)-[:ROUTED_TO]->(agent:Agent)
WITH agent, avg(agent.confidence) AS avgConfidence
RETURN agent, avgConfidence
ORDER BY avgConfidence DESC;
```

Query 4: Get Agents Matching TaskType Requirements

```
MATCH (taskType:TaskType)-[:REQUIRES_CAPABILITY]->(cap:Capability),
      (agent:Agent)-[:HAS_CAPABILITY]->(cap)
RETURN DISTINCT agent;
```

Query 5: Retrieve Fallback Agent When Confidence Is Low

```
MATCH (agent:Agent)-[:FALLBACK_AGENT]->(fallback:Agent)
RETURN fallback;
```

6. Multi-Agent Workflow Architecture (Revised)

Multi-Agent Workflow Architecture (Revised) Instructor recommended clearer CrewAI-style decomposition.

6.1 QueryAnalyzer Agent

- Parses user query
- Extracts entities using GLiNER or GPT-4o-mini
- Produces structured representation:

```
{
  "task_type": "code review",
  "complexity": 0.6,
  "domain": "technical",
  "output_format": "summary"
}
```

6.2 KnowledgeGraphQuery Agent

- Builds SPARQL/Cypher queries dynamically
- Retrieves candidate agents
- Ranks using:
- capabilityLevel
- historical performance
- domain alignment

6.3 RoutingDecision Agent

- Chooses optimal agent
- Generates routing rationale
- Computes confidence score
- Handles tie-breaking logic

6.4 FeedbackCollector Agent

- Monitors success/failure
- Updates capabilityLevel, historicalAccuracy
- Allows the graph to improve over time

---

7. Entity Extraction Strategy (Revised)

Instructor asked for a firm commitment.

Final Choice: GLiNER

Reasons: - Zero-shot extraction - High accuracy for custom entities - Fast inference - No need for large labeled datasets

Entities extracted: - TASK_TYPE - COMPLEXITY_LEVEL - DOMAIN - OUTPUT_FORMAT

Mapping Example:

```
"code debugging" → ex:CodeDebuggingTask
"complex" → complexityLevel 0.8
```

---

8. Agent Catalog (Demo Scope)

Instructor recommended 4–5 prototype agents.

1. WebSearchAgent

2. CodeAnalysisAgent

3. SummarizationAgent

4. DataVisualizationAgent

5. PerplexityFallbackAgent

Each has: - capabilities - capabilityLevel - fallback relationships - sample historical performance

---

9. Demo Scenarios (Instructor Recommendation)

Scenario 1: "Find the latest research on LLM pruning."

Router → WebSearchAgent

Scenario 2: "Debug this Python snippet."

Router → CodeAnalysisAgent

Scenario 3: "Summarize this contract."

Router → SummarizationAgent

Extra:

Scenario 4: Low confidence

Router → PerplexityFallbackAgent

---

10. Visualization & Explainability

Neo4j Bloom or GraphXR will show: - how agents relate - fallback chains - why the router selected an agent

Example explanation:

> "Selected CodeAnalysisAgent because it has CodeUnderstanding capability (0.85), matches TaskType CodeDebuggingTask, and has the highest historical confidence (0.82)."

---

11. Final Implementation Plan

Week 1

- Build RDF schema
- Implement full SHACL
- Create Neo4j graph with sample nodes

Week 2

- Implement GLiNER extraction
- Implement QueryAnalyzer, KGQuery agents

Week 3

- Implement router logic + confidence scoring
- Add fallback handling

Week 4

- Build demo UI
- Add feedback loop
- Prepare elevator pitch & visualizations

---

12. Elevator Pitch (Updated)

> "The Smart Agentic Router solves the *agent selection problem* by using knowledge graphs to route user queries to the best specialized AI agent. Unlike simple keyword-based systems, our router uses semantic reasoning, agent profiling, and performance-based learning to deliver accurate, explainable, and continuously improving routing decisions."

End of Updated Proposal