Software Requirements Specification (SRS)

Towards Lightweight Routing: Adaptive and Explainable Orchestration of Models and Agents

1 Introduction

1.1 Purpose

The system aims to develop a lightweight, adaptive, and explainable routing framework that operates locally without relying on external cloud APIs. Its goal is to efficiently orchestrate multiple language models (LLMs) and specialized agents directly on edge or low-resource computing environments (e.g., personal laptops, workstations, institutional servers), ensuring data privacy, offline functionality, and resource efficient AI inference.

1.2 Scope

The system focuses on developing an intelligent query routing mechanism that classifies and directs user inputs to the most appropriate local model or agent. Key features include:

- **Adaptive Learning:** The router updates its strategy based on local feedback and performance metrics.
- **Explainability:** Each routing decision is accompanied by an interpretable explanation visible to users.
- **Cost and Performance Awareness:** Balances response quality with computational efficiency for edge hardware.
- **Local Orchestration:** All models and agents are hosted and executed locally; no cloud dependency.
- **Visualization Dashboard:** Interactive display of routing patterns, cost-performance metrics, and decision rationales.

2 High-Level Description

2.1 Product Perspective

The routing system acts as an intermediate decision layer between the user and multiple AI components (models and agents). When a user submits a query, the router analyzes it, determines complexity and intent, and assigns it to:

- A small local model for simple tasks.
- A specialized agent for domain-specific tasks (e.g., code generation, retrieval).
- A larger local model for complex reasoning or multi-step queries.

Architecture components include the Routing Module, Adaptive Learning Component, Explainability Engine, and Dashboard Interface. All components run locally on edge hardware to ensure privacy and reduced latency.

2.2 User Training

Users will be provided with tutorials: Developers/Admins will receive guidelines on adding/removing models and retraining the router; End Users will have instructions on querying and interpreting explanations via the dashboard.

2.3 System Constraints

- Must operate on standard edge devices (8–16GB RAM, mid-level CPU/GPU).
- Uses only local open-source models; no external API calls.
- Initial support for up to ten models/agents running concurrently.
- Adaptive learning runs locally without requiring internet access.

2.4 Assumptions and Dependencies

- Users have hardware capable of running small to medium LLMs locally.
- Router training dataset is prepared and stored locally.
- System depends on local model runtimes (e.g., Llama.cpp, Ollama, Transformers local inference tools).
- Hybrid cloud integration is optional and not part of the initial deliverable.

3 Specific Requirements

3.1 Integration and Performance Requirements

3.1.1 User Interfaces

- **Dashboard:** Shows routed queries, chosen model/agent, performance metrics, and explanation.
- **Admin Interface:** Manage models/agents, update datasets, trigger retraining, and view logs.

3.1.2 Hardware Interfaces

- Compatible with laptops, desktops, and local servers.
- Optional GPU acceleration supported for inference; system works without GPU (slower).

3.1.3 Software Interfaces

- Integration with locally hosted models and agents through direct execution (in-process calls, local subprocesses, or inter-process communication like sockets).
- Each model or agent runs as a local module or process; the router invokes them without external network calls.
- Local dataset interface for continuous retraining and evaluation.
- Local storage for query logs, model metadata, and performance statistics.

3.1.4 Communication Interfaces

- Internal communication via local REST endpoints, UNIX sockets, or command-line interfaces.
- No external internet communication required unless explicitly enabled for optional updates.

3.2 Functional Requirements

3.2.1 Query Classification and Routing

- Analyze input queries to determine agent vs LLM handling.
- Use lightweight feature extraction and a trained router model for classification.
- Record routing decisions and outputs for feedback and evaluation.

3.2.2 Adaptive Learning

- Update routing parameters using local feedback signals, such as user approval metrics or automatic evaluation scores.
- Automatic escalation: If a model's response fails local validation, re-route to a higher-capacity model and log the event.

3.2.3 Explainability Layer

- Provide natural language justifications for routing choices (e.g., "Routed to Math model due to detected math expressions and high complexity").
- Visualize confidence scores and cost-performance trade-offs on the dashboard.

3.2.4 Model and Agent Management

- Admins can add/remove local models and agents via the Admin Interface.
- New models/agents are auto-registered into the routing pool with metadata (cost, capacity, domain).
- Maintain logs of usage, latency, accuracy, and cost metrics per model/agent.

3.2.5 Dashboard and Visualization

- Query history, routing trends, Pareto front visualizations, and filterable views by model, time range, and domain.
- Exportable logs and reports for evaluation and defense.

3.2.6 Routing Decision Logic

The core functionality of the system lies in its intelligent router that determines which model or agent should handle each incoming query. The decision is made based on a combination of query-level features and system-level constraints.

- **Feature Extraction:** Each query is analyzed to extract linguistic and structural features (e.g., token length, reasoning indicators, domain-specific keywords).
- **Complexity Estimation:** The router estimates the query's time and space complexity, predicting whether it requires high reasoning power or large memory/context windows.
- **Model Profiling:** Each model or agent stores metadata such as processing speed, memory footprint, accuracy rating, and domain specialization.
- **Decision Function:** The router evaluates all models against the query's estimated complexity and selects the optimal model balancing accuracy, latency, and cost.
- **Adaptive Feedback:** After execution, the router records the model's output quality and updates its internal weights to refine future routing decisions.

This ensures that simple queries are routed to smaller, faster models, while complex tasks are escalated to larger or specialized models, optimizing trade-offs between speed, cost, and accuracy.

4 Non-Functional Requirements

4.1 Security

- All data, models, and logs are stored locally; sensitive data encrypted at rest.
- No cloud storage or external API keys required by default.
- Access control for Admin Interface via local authentication.

4.2 Reliability

- Target 99% local uptime for core routing services.
- Automatic process monitoring and restart for failed model processes.
- Fallback routing to alternate models on failures.

4.3 Availability

- Fully functional offline; optional online updates when explicitly enabled.

4.4 Scalability

- Modular architecture allowing additional local models/agents to be plugged in.
- Designed for resource-constrained scaling on edge devices.

4.5 Maintainability

- Modular codebase, documented APIs, and unit tests for core components.
- Clear developer guides for retraining and adding models/agents.

4.6 Usability

- User-friendly dashboard with labeled metrics, explanations, and color-coded indicators for confidence and cost.

4.7 Performance

- The system should deliver smooth and responsive query routing suitable for real-time use on local machines.

- The router will be optimized for minimal processing overhead, ensuring quick decision-making between models and agents.

- The dashboard will provide fast and seamless interaction, even as new data is added or adaptive learning occurs.

- The overall design will remain lightweight and efficient, capable of running effectively on devices with moderate computing resources (CPU/GPU).