Totem Interactive

ML Developer Assignment

Dynamic LLM Router & RAG System

Design a **plug-and-play** solution that **routes** user requests to different LLM providers based on **request type** (e.g., “technical,” “creative,” “general”). In parallel, it **asynchronously** queries a **low-fidelity model** to ensure there’s always a fallback response if the primary model fails. The system must also **log all interactions** and **dynamically update** each LLM’s priority based on real-world performance. Additionally, allow a mechanism to **search for the “best” LLaMA model** for a given use case and append it to the active configuration.

Core Objectives

1. **Request Categorization & LLM Selection**
    - Classify each user query (e.g., “technical,” “creative,” “general”) using basic NLP or a rule-based approach.
    - Map each category to a **preferred** LLM (e.g., a creative LLM for “creative,” a more factual LLM for “technical,” etc.).
2. **Asynchronous Fallback Model**
    - In parallel with the “preferred” model call, **fire a request** to a **lower-fidelity** (or cheaper) model.
    - If the preferred LLM times out or fails, **immediately return** the fallback model’s response.
    - If the preferred LLM’s response arrives successfully after the fallback, you can update the conversation or just log it for reference.
3. **Performance-Based Dynamic Priority**
    - **Log** all responses (success, time taken, cost, etc.).
    - Adjust the **priority** of each LLM **per category** based on observed performance (e.g., if an LLM consistently fails for “technical,” move it down in the priority list).
4. **RAG Application (Low-Level) & Model Extension**

- Treat the fallback model as a **Retrieval-Augmented Generation** (RAG) style or a simple knowledge-based model to ensure minimal coverage.
- Implement a **“search for the best LLaMA model”** step that can discover new LLaMA variants (or other models) for a category, then **append** them to the config if they seem promising.
- This search could be a mock or partial implementation that simulates calling an API or scanning a repository for the “best” LLaMA model.

5. **Logging & Dynamic Config Updates**
   - Store logs with details of each request:
     - **Category** assigned
     - **Preferred LLM** used
     - **Fallback LLM** used (if any)
     - **Response time**, **Success/Fail**, and **Cost** (if available).
   - Based on logs, **dynamically** reassign the best model for each category (e.g., “technical” => Model B if Model A is failing).
6. **Demo & Real-World Testing**
   - Prepare a small **demo UI** or CLI to show how the routing works for different query types.
   - The final step involves **replacing** a mock LLaMA setup with a **real LLaMA API key** (provided later in-person) and testing **debugging** or **live performance** in your environment.

Mandatory Features

1. **Categorization Logic**
   - At least a **simple** classification approach to label queries by topic or style.

2. **Asynchronous Requests**
    - Fire calls to multiple models simultaneously or in parallel threads/processes.
3. **Fallback Mechanism**
    - Immediately serve the fallback model's output if the primary fails or lags beyond a set timeout.
4. **Dynamic Priority Updates**
    - Re-rank LLM providers over time using logs.

Deliverables

1. **Source Code (Microservice or Library)**
    - A **well-structured** repository (e.g., GitHub) with a **clear README**.
    - Include **installation** and **run** steps (Docker or local environment).
2. **Configuration & Categorization Files**
    - A config for LLM providers (endpoints, cost, priority per category).
    - Basic rules or an ML model for categorizing requests (technical, creative, etc.).
3. **Logging & Dynamic Updates**
    - Show logs for each request in a structured format (JSON, database, or console).
    - Demonstrate how the system reorders LLM priorities over time based on logs.
4. **Search & Append Mechanism**
    - A **mock or minimal** function that "searches" for a new LLaMA variant and appends it to the config if it passes some criteria.

   - This can be partial, but show the logic of how it would integrate.

5. **Demo UI**

   - A **simple interface** (web page or CLI) where you can enter a prompt, see which model(s) got called, the fallback model if used, and the final response.

Evaluation Criteria

1. **Technical Implementation (40%)**

   - Proper asynchronous calls, fallback logic, dynamic priority re-ranking.

   - Quality of request categorization approach and overall code structure.

2. **Resilience & Logging (30%)**

   - Robust error/timeout handling for the main vs. fallback model calls.

   - Thorough logs that enable dynamic updates to the model priority.

3. **Search & Append Feature (20%)**

   - Creativity and clarity in simulating or partially implementing a “search” for new LLaMA variants.

   - Proper integration into the main config so the new model can be used if it’s better for a certain category.

4. **Demo & Usability (10%)**

   - A straightforward UI/CLI to show how the system reacts to different query types.

   - Clarity in how developers can integrate or extend this system.

Submission Guidelines

Totem Interactive

ML Developer Assignment

1. **GitHub Repository**
   - Name it, for example, `Dynamic-LLM-Router-{YourName}`.
   - Include separate folders for core code (`/src` or `/app`), config files, logs (if needed), and your demo UI.
2. **README.md**
   - Detail setup instructions, dependencies, usage examples.
   - Explain assumptions, limitations, and any special instructions for running asynchronous tasks.
3. **Commit & Push**
   - Ensure version control with clear commit messages.
   - Provide instructions on how to run the final service or test script.
4. **Live Integration Step** (Optional but Encouraged)
   - After the baseline is done, be prepared to **replace** your mock LLaMA integration with a **real** LLaMA API key (provided in person) and show debugging or performance testing.

Submit your GitHub repo URL and any additional notes (e.g., how to replicate the search & append feature if it requires specific external services). We'll review your code and potentially test it with our own keys to see how well it adapts in real time.

Objective:
Create an **LLM router** that intelligently **assigns requests** by category, uses a **fallback** model asynchronously, **logs everything**, and **updates priorities** dynamically based on performance. Demonstrate a method to **search and add** new LLaMA models, ensuring your system remains **flexible and scalable** for evolving AI needs.

Notes

Totem Interactive

ML Developer Assignment

- The implementation approach is open-ended; focus on fulfilling the core requirements.
- Prioritize the quality of analysis logic over interface aesthetics.
- Feel free to include additional features if they enhance functionality or user experience.

Submission

Submit the following files to career@toteminteractive.in / Totemistaken@gmail.com / Shoeb@toteminteractive.in / arjungujar@toteminteractive.in:

1. Create a GitHub repository for your assignment solution
   - Provide a descriptive name (e.g., "{Name}-assignment-solution")
   - Create a folder for your code (e.g., src or code)
   - Add solution files (e.g., .swift, .py, .js in the coding language of your choice)
   - Use Git for version control and commit changes
2. Create a README.md file in the root directory
   - Explain implementation details and approach
   - Provide usage instructions
   - Mention assumptions, limitations, and any identified edge cases
   - List dependencies and additional notes
3. Commit and push all changes to the remote repository
4. Submit the GitHub repository URL to the designated contact person