Report on LLM usage in the message dispatch tool

Introduction

This report outlines the role of Large Language Models (LLMs) within the project, detailing their implementation, relevance, and key considerations. The project enables users to define system and user messages, dispatch them to multiple LLMs via OLLAMA, and store responses into a MongoDB database.

LLMs' role in the project

The project serves as an LLM message dispatch tool, which allows users to interact with multiple locally installed LLMs. Users define input messages through a React-based front-end, which are then processed by OLLAMA. The selected LLMs generate responses, which are shown on the UI and stored for further analysis into a MongoDB database.

LLMs used by the project group

- Marco-o1:latest
    - The system prompt must be very explicit for the LLM to take it into consideration.
- Llama3.2:3b
    - Working very similarly to Marco-o1

Implementation of LLMs

- **Front-End Integration:** The user inputs messages via a React front-end. Responses are shown model-specifically on the same user interface.
- **Back-End Processing:** The messages are forwarded to OLLAMA, which serves as an interface and a platform for running LLMs.
- **Containerization:** OLLAMA is containerized using Docker, which helps with portability and ease of deployment.
- **Database Storage:** The generated responses are stored in a MongoDB database, from which retrieval and further analysis can be done.

This implementation allows users to dynamically select LLMs without requiring predefined models, which enhances the project's adaptability.

Relevancy of RAG, fine-tuning, etc.

Based on the project's objectives and the assignment requirements, techniques such as Retrieval-Augmented Generation (RAG), fine-tuning, and prompt engineering were deemed unnecessary for this project. The primary focus was on dispatching messages

and collecting the responses rather than optimizing LLM outputs through advanced customization.