Auto model switching Logic

Semantic analysis of user message and comparing them with embeddings of each model

This approach is highly efficient and accurate, in this method we perform 3 steps

STEP 1 => Make an embedding vector of each description of a given model, i.e. Suppose I can give a description example of Gemini as Find me a spam email, then we run the script generateEmbeddings.js which generates a vector corresponding to Find me a spam email and stores it within the embeddingPhrases.json itself.

NOTE:- the embeddings are generated using a local model mistral which can be easily set up with Ollama on the corresponding hosting server app. We can also use ChatGPT or Gemini to create the embeddings, but we will have to pay for it therefore it's much more cost effective to set up a small local model which effectively reduces out costs to zero

STEP 2 => When the user inputs a message with auto mode on, the message is again converted into a embedded vector using Ollama.embed( ) and since this is a small local model, this effective costs us zero tokens although some CPU computational power may be required

STEP 3 => We loop through all the embeddings generated in step 1 of various descriptions across all models and compare the user Message embedding with them, We compare by calculating the embedding whose vector has the least angle difference i.e. maximum cos(theta) using the angleBetween() function, and we output the model corresponding to the maximum cos(theta) between those 2 vectors(cos theta is calculated by (A.B)/(|A|*|B|)

How I have optimised token usage in my LLM API calls

1. Optimisation in model `auto` selection logic
   => Here i have effectively reduced the `token count` to zero as we are using a small local model for embedding

2. Optimisation while implementing Contextual Memory in the same chat
   => Instead of passing all the previous messages to the model to handle the contextual memory, for each chat I have defined a key called `summary` which stores all the chat info in short way including only the key points. And each time I make a call to API I send `summary` of conversation before it and not all the messages, this optimises my token count further

3. Optimisation while generating `summary` after each message in conversation
   => You might think that but for updating the conversation summary after each conversation you might have to call API 2 times one for generating user answer and one for updating summary based of that answer. **BUT NO,** We are generating summery using the same small `Ollama mistral` model which we used during `auto switching logic` as these models can also generate short summaries of long responses. Therefore, after every message it takes the `previous summary`, `current user question` and `current LLM response` to effectively generate the `Updated summary` including key details about what was before and what was after.