This is the Documentation file explaining

1) Auto Model switching Logic

2) Why my logic is effective

3) How i have optimised token usage in my API calls to the LLM

1 Auto model switching Logic

I have developed 3 methods for Auto model selection , the developer can choose any. However the method i think is most effective is METHOD 3

Method 1 : `Asking another LLM to decide which LLM to use for that particular user question`
explanation
=> “This method takes the user message and forwards it to a LLM (NOTE this LLM can be choosed by developer whose cost is the least) in my case it is e.g ChatGPT”

=> "This model takes the messase and a `system` prompt of giving back a model name from an enum of values `["chatgpt", "gemini", "gemma", "mistral", "claude"]` based on the user `message`"

Method 2 : `Manual Keyword maching`
explanation
=> “This is a very less effective but the fastest method, we simply match the keywords exactly with the user input”

=> “We have a set of keyword phrases for each model in `keywords.js` so when user types `Translate the phrase ...` our algorithm convers the user message to lowercase and matches the keyword `translate` with `gemma` list of kewords and hence gemma is used”

=> “Note this approach is very naive and less effectie as it doesn’t handle user typos and al possible keywords”

MOST EFFECTIVE METHOD

Method 2 : `Semantic analysis of user message and comparing then with embeddings of each model`
explanation
=> “This approach is highly efficient and accurate, In this method we perform 3 steps”

=> *STEP 1* => “Make a `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 corresponsing 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 effecrtive to set up a small local model which effectively reduces out consts 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` altho some CPU computational power may be required

=> *STEP 3* => We loop through all the embeddings generated in step 1 of variaous descriptions across all models and compare the userMessage embedding with them , We compare by calculating the embedding whose vector has the `least anngle 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|))

THEREFORE in my project you can change in .env file which DEFAULT_MODEL_SELECTION to use , you can set it to `semantic` for method 3, `manual` for mehthod 2 and `chatgpt` for method 1 using chatgpt as a model

How i have optimised token usage in my LLM API calls

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

2. Optimisation while implimenting 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 chatinfo 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 cal API 2 times one for generating user answer and one for updating summary based of that andwer.**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 messae 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.