Kyndryl LLM: Save User Development time with Private Data-Integrated Generative AI

Arvind Balaji, Jack Boller, Yonghwan Kim, Anh Nguyen, Phu (Jack) Nguyen

## Introduction

In the vast expanse of the internet, the LLM is trained using public data, which can sometimes be filled with irrelevant or low-quality information. To address this, integrating private data can be a game-changer. It allows for the fine-tuning of the model, ensuring more precise and relevant responses. The following document will contain information about user profile, data provision, system performance, and application flow.

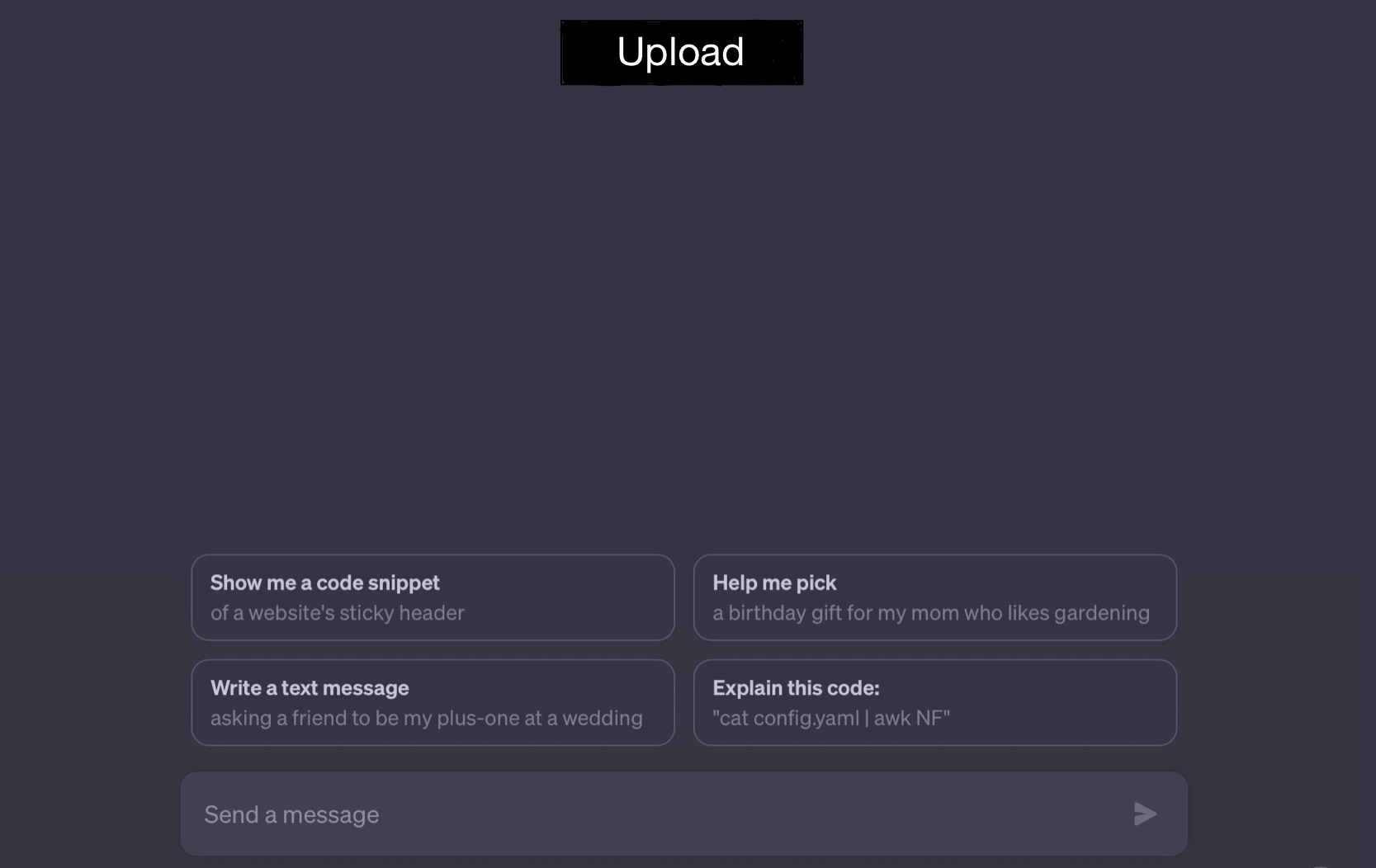
## User Profile

The main users of this system rely on the private data from the API documentation. While these individuals usually have a good grasp of the company's offerings, they might not be experts in every specialized area. Additionally, the influence of public data can sometimes skew the AI agent's responses. Every day, they address a wide array of questions, from simple fixes to intricate technical challenges, and even craft new use cases. By incorporating a generative AI system that accesses the private API documentation, these agents can deliver quicker and more precise answers, enhancing the overall user experience.

## Data Provision

## 

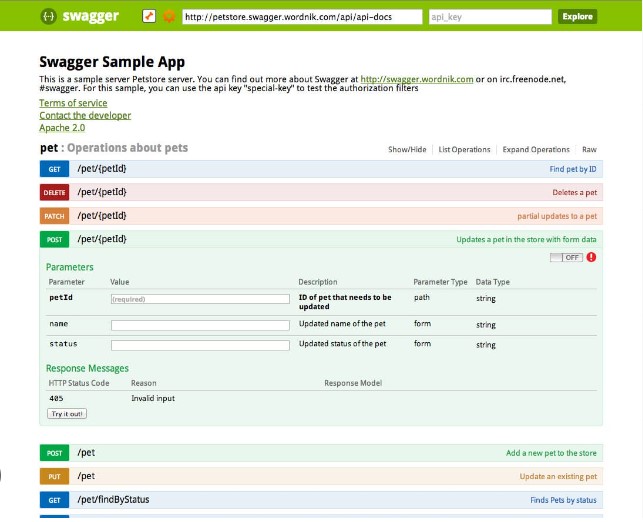
Supported Data Types



UI Prototype

The user, in this case, the support agent, provides the system with a user's query. This query can be in the form of a text question, a product code, or even a description of an issue. Additionally, the organization feeds the system with curated private data. This data can originate from:

1. Swagger API documentation
2. Development FAQ
3. .DOCX or similar types format
4. URL for Documentation API



Example documentation

## Expected System Performance

Upon receiving a query, the user expects the system to:

1. Quickly parse and understand the context of the question.
2. Search through the integrated private data (embeddings vector database) to find the most relevant and accurate information.
3. Generate a coherent, concise, and accurate response that addresses the user's query.
4. Offer suggestions or further reading, if necessary and available, from the private data sources.

## Application flow

System Diagram

1. Query Input: The support agent inputs the user's query into the system through UI interface.
2. Data Processing: The system processes various file types from the private data sources, standardizing them into a uniform format suitable for AI interpretation such as prompts and results.
3. Tokenization: The processed data is tokenized, breaking it down into smaller chunks that the AI can understand and analyze.
4. Transformation into OpenAI Embeddings: Using Azure Open AI, the tokenized data is transformed into embeddings. These embeddings are essentially numerical representations of the data, making it easier for the AI to search and match relevant information.
5. Storage in Pinecone Vector Database: The embeddings are stored in the Pinecone Vector database, a specialized database designed for high-speed retrieval of such data.
6. Generative AI Response Generation: When a query is inputted, the system searches through the embeddings to find the most relevant information. The generative AI then crafts a response based on this information, ensuring it's coherent and contextually accurate.
7. Output to User: The generated response is presented to the support agent, who can then relay the information to the user.