Indexify.AI
Improving Chat Bot RAG

MEET THE TEAM

AVIV BAREL

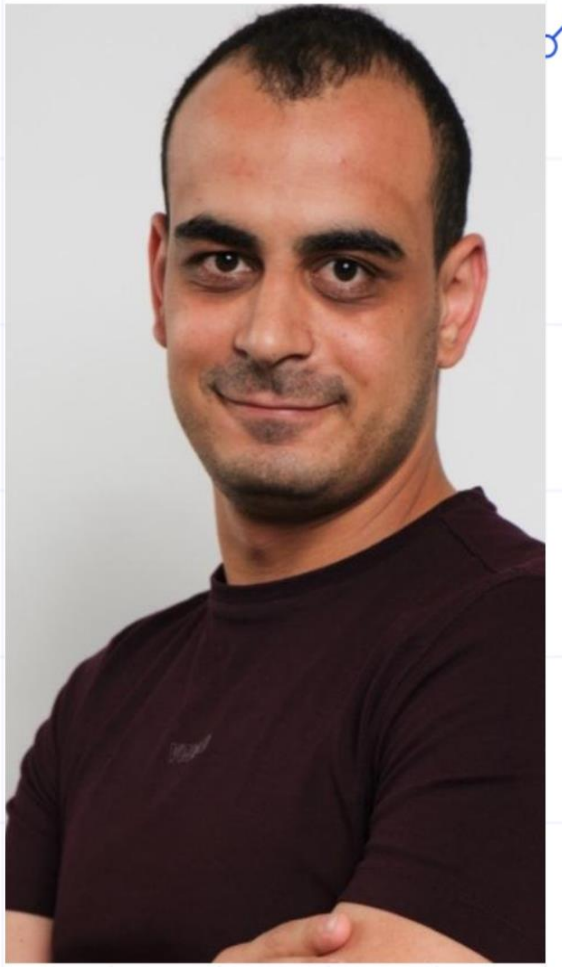
Founder & CEO

LIHU ZUR

CTO

BASEL AMIN

Senior Data Scientist



ALI ABU LIL

Senior ML Engineer

OUR MISSION

Indexing your data - for your LLM

TEAM COMPOSITION

A team of four passionate data scientists working collaboratively.

MISSION STATEMENT

Enhancing and fine-tuning a financial chatbot powered by LLMs to deliver more accurate and personalized financial recommendations.

PROJECT FOCUS

Centered on achieving enhanced relevancy and improved accuracy in results.

TECHNIQUES USED

Utilizing hierarchical indices and summarization techniques for efficiency.

GOAL

Improving retrieval-augmented generation (RAG) for smarter responses.

CURRENT FLOW

DATA IS STREAMING FROM ALPACA FINANCIAL NEWS

Financial news is fetched using a batch API and an online WebSocket for real-time updates.

DOCUMENTS ARE BEING SAVED IN QDRANT

Qdrant, a vector database, stores the data as embeddings for efficient retrieval.

LLM FINE-TUNED WITH QLORA AND REGISTERED IN COMET ML

The LLM, Falcon 7B model from Hugging Face, is fine-tuned using QLoRA and saved in the model registry in Comet ML.

.QUERYING THE MODEL USING LANGCHAIN

A chain is built where the most relevant documents are retrieved from Qdrant using KNN. These documents, along with the user query, are provided as context to the LLM for generating responses.

Issues with Current Data

Irrelevant Data

Some of the retrieved documents are occasionally irrelevant. As a result, any data pulled from those documents can confuse the LLM by providing incomplete or misleading context.

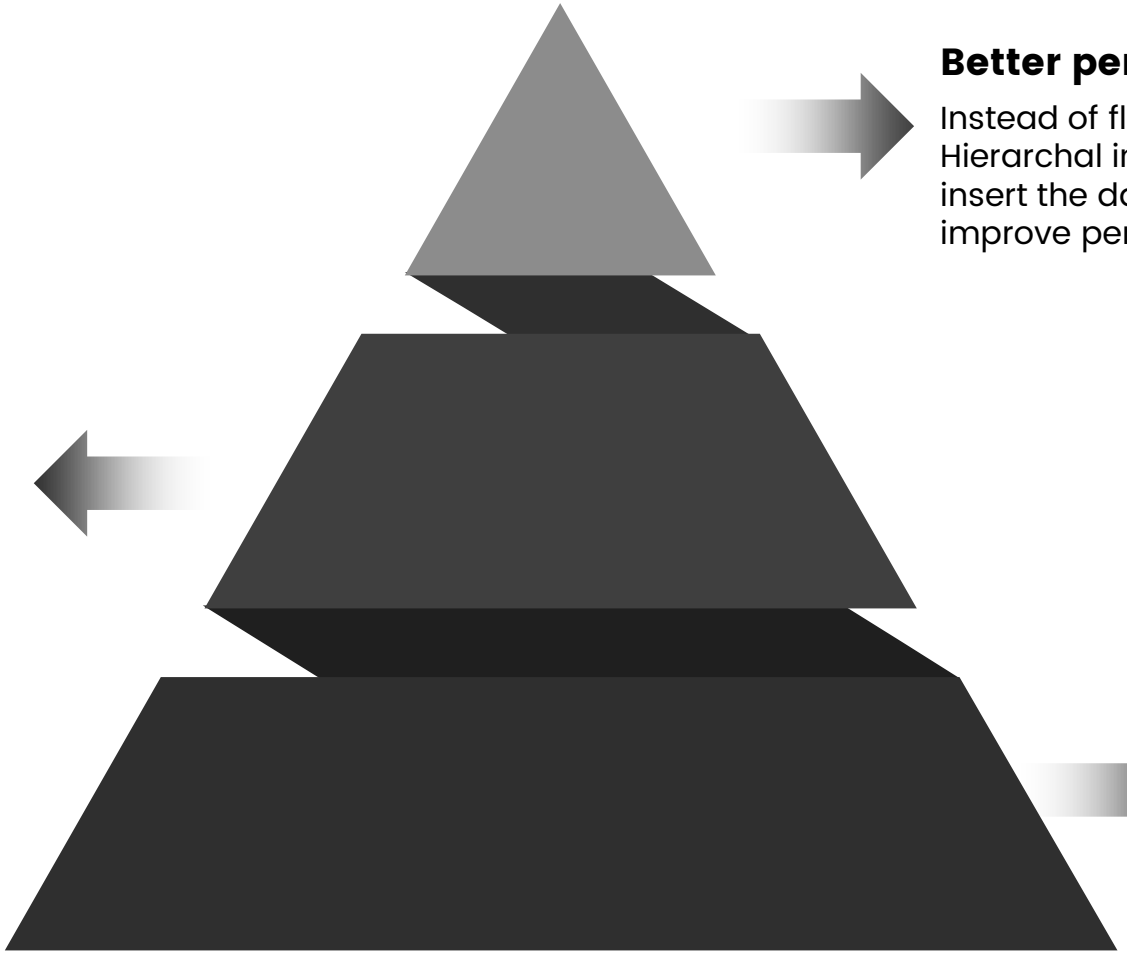
01

Missing Summaries

The LLM only includes summaries of each fetched document in its context. However, if a document has no summary, it contributes nothing to the final answer

02

Objectives for Improvement



Better performance

Instead of flat hierarchy, using Hierarchal indexing we can search and insert the data faster to the DB and thus improve performance

More Concise Answers

Enhance the clarity and precision of responses.
Minimize unnecessary details.
Ensure answers are concise and focused on delivering the most relevant information.

Better Scalability

Larger datasets become more manageable when organized hierarchically.

WE ARE SUGGESTING :
HIERARCHCY INDICES AND SUMMARIZATION

Streaming Pipeline - Document Classification

Identifying the Sector

For each document, query GPT 4o- mini to classify it into a SectorThe broad industry/field to which the document belongs (e.g., Finance, Healthcare, Technology).

Identifying the Subject

Given the sector and the document, query GPT again to classify it into a subject under the sectorThe specific company/subject/product mentioned in the document (e.g., Microsoft, Climate Change, Tesla).

Identifying the Event Type

Given the sector, subject, and document, query GPT again to classify it into the event type under the given sector and subjectThe type of event/activity described in the document (e.g., financial report, scientific discovery, product launch, etc.).

Streaming Pipeline – Query with GPT

Problem

- Chat-GPT returns extra words that makes categorizing more difficult.
- It Must not respond with “none of them” or “not sure.”

Strict Output Requirements

Restricting GPT to return only the category name with no extra text. Must not respond with “none of them” or “not sure.” Instead, it must choose from the current categories or propose a new category if none of the given options fit.

Streaming Pipeline – Query with GPT

Output Specifications

GPT tends to misinterpret the layers if not explained properly. It might label a sector as an event or vice versa, also it maybe can have bad classifications, changing the format, etc.

Addressing Classification Challenges

Convincing GPT to avoid its usual explanatory style and produce a single, direct answer.
Ensuring it follows the hierarchy in the correct order (Sector → Subject → Event Type), also giving examples for better understanding.

Streaming Pipeline – Prompt with GPT

```python
def classify_with_gpt(self, text: str, options: List[str], level: str, sector: Optional[str] = None, subject: Optional[str] = None) -> str:
    system_prompt = f"""
        You are tasked with classifying a document into the following three categories:

        ### 1. **Sector**:
        The broad industry or field to which the document belongs (e.g., Finance, Healthcare, Technology).
        - **Explanation**: The **Sector** is the broadest classification. For instance, if the document is about healthcare services or innovations in the medical field,
         it would fall under **Healthcare**. If it's about technology products or software development, it would fall under **Technology**. Please pick the sector that fits best based on the text.

        ### 2. **Company/Subject**:
        The specific company or subject mentioned in the document (e.g., Google, Tesla, artificial intelligence, climate change).
        - **Explanation**: The **Company/Subject** level focuses on the specific company or subject mentioned. For example, if the document mentions a new technology by **Apple** or discusses **Artificial Intelligence**,
         the response should reflect that. If the subject doesn't match the options, suggest a fitting one. You can classify topics like "climate change" under the **Subject** category even if it isn't a company.

        ### 3. **Event Type**:
        The type of event or activity described in the document (e.g., merger, financial report, product launch, acquisition, scientific discovery).
        - **Explanation**: The **Event Type** categorizes what the document describes in terms of events or activities. For example, if the document talks about a company merger, it should be classified under **Merger**.
        If it's about a product release by **Apple**, it should be classified as a **Product Launch**. If no event type matches the options, suggest one based on the document's context.
                """
    user_prompt=""
    # Building the prompt based on the level
    if level == "subject":
        user_prompt += (
            f"you need to decide which subject the following text belongs under the sector '{sector}':\n\n"
        )
    elif level == "event type":
        user_prompt += (
            f"Based on the following text, decide which event type it belongs to under the sector '{sector}' and subject '{subject}':\n\n"
        )
    else:
        user_prompt += (
            f"Based on the following text, decide which {level} it belongs to:\n\n"
        )
    user_prompt += f"Text: {text}\n\n"
    user_prompt += f"Options: {', '.join(options)}\n\n"
    user_prompt += (
        "If the list of options is empty, or none of the options seem appropriate for any of the categories, "
        "**suggest an appropriate one** based on the content of the query. "
        "Your suggestions should be **specific and relevant** to the content. "
        "**Do not reply **neither of the options** or **none of them** or anything of the sort! this is not valid answer. "
        f"Always provide an answer, even if it means suggesting a new category that fits better.\nThe answer must be only the name of the {level} without any garbage!")
```

Streaming Pipeline – Hierarchy construction

Construct the Full Hierarchy of the File

01

For each sector a new node in the tree is being created , and inside it can be found all subjects, and inside each one of them the event types

02

Attaching each “triplet” (sector_subject_event-type) to the document payload and save in Qdrant, which we will use in the inference pipeline

03

Save the full hierarchy into a JSON fileStore, the full hierarchy extracted from all files in a tree- based structure

Inference pipeline - Query Classification and Context Extraction

Classify the Query

Using the data structure saved inside our JSON, we classify the query into sector, subjects, and event types.
Allow more than one classification to extract various yet concise contexts.

Top_k Approach

Limit the query classifications to 1 topic, 3 subjects under the topic, and 5 event types under the sector and subject.
Group all the extracted documents, sort them by score (descending order), and finally choose top 5 documents as context.

Context Addition

Utilize Document Summaries

If the summary existed, use it. Otherwise, call GPT to summarize the text for us.

Final Context Compilation

The final context for the LLM will be the concatenation of all summaries extracted from each of the matched documents (partially GPT-generated, partially originally existent).

Prompt Classification for User's Query

Key Challenges

- **Purpose**:
  Classify a user's query at different levels (Sector, Subject, or Event Type) and impose very strict rules on how many and which options can be returned.

- Dynamic Instructions Depending on whether targeting a Subject, Event Type, or another classification, the prompt text changes.

- In Contrast to the Streaming pipeline prompt , We enforce GPT to chose from the given options without creating new ones

- Same as Streaming pipeline:
  The model must never say "none of them" , no additional Explanation to the desired output

EXAMPLE : SHOULD I INVEST IN 'MICROSOFT'?

01

FINDING SECTOR

Technology

02

FINDING COMAPNY/ SUBJECT

Microsoft

03

FINDING EVENT TYPE

Stock_price

04

MOST RELEVANT DOCUMENT

As of January 18, 2025, Microsoft Corporation's (MSFT) stock is trading at $429.03, reflecting a 0.99% increase from the previous close.

IMPROVING THE RAG RESULTS EXAMPLE

01

ABOUT ME

I'm 26 years old , software engineer, looking to invest my money in the US stock market

02

CONTEXT

As of January 18, 2025, Microsoft Corporation's (MSFT) stock is trading at $429.03, reflecting a 0.99% increase from the previous close.

03

USER QUERY

Should I invest in Microsoft ?

Summary Precision and Tone with GPT

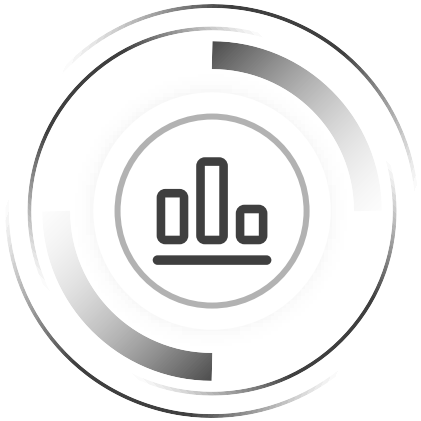
Background

We needed a concise overview of financial documents but wanted it to feel like a direct answer rather than a typical summary.
Previous attempts ended up as bullet points with no real summary or sounded too formal.

Goal

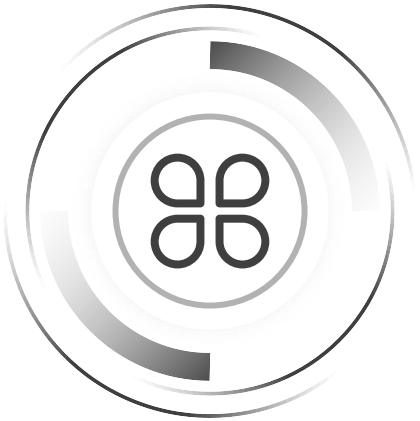
Summarize the text but avoid “This document says…” or “The text discusses…” language.
Striking the right balance—keeping it both concise and conversational—was much harder than it first appeared.

Addressing Technical and Practical Issues



GPT Responses

Sometimes GPT did not fulfill the task correctly.
Solved by engineering the query for each case using a trial and error method.



Preserving Hierarchical Order

Qdrant does not support hierarchical storing so we used the JSON approach instead.

Collection Limits

Initially wanted to create a collection for each triplet of sector- subject- event type. Qdrant limits the number of collections used the original collection and stored the triplet inside payload of each document instead.

Improved metrics

Metric	Baseline Bot	Improved Bot	Difference
Answer similarity	0.52	0.64	0.12
Faithfulness	0.293204	0.5392	0.2492

Improvement in Answer Similarity:

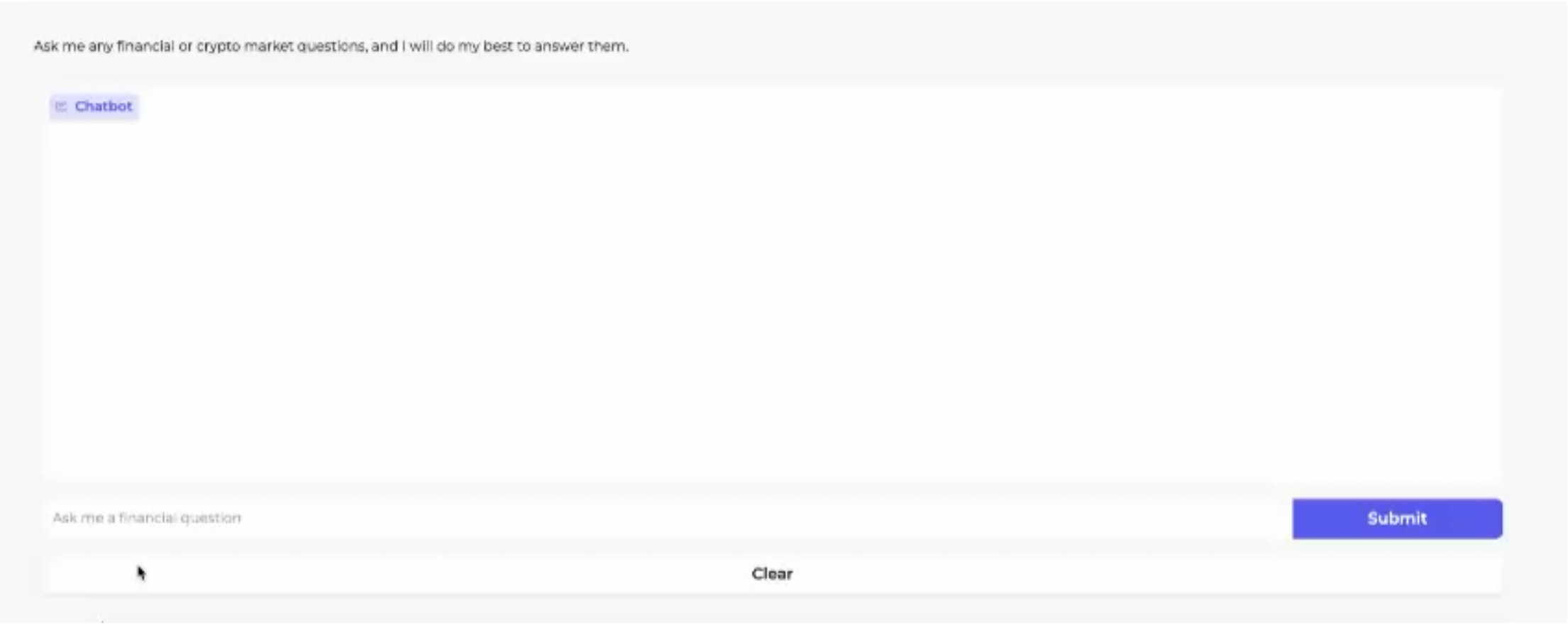
• Hierarchical Indices: Organized data contextually reduces noise, making retrieval more precise by narrowing down to relevant subsets.

• Summarization: Condenses long documents into focused, relevant points, aligning the retrieved content closer to the query.

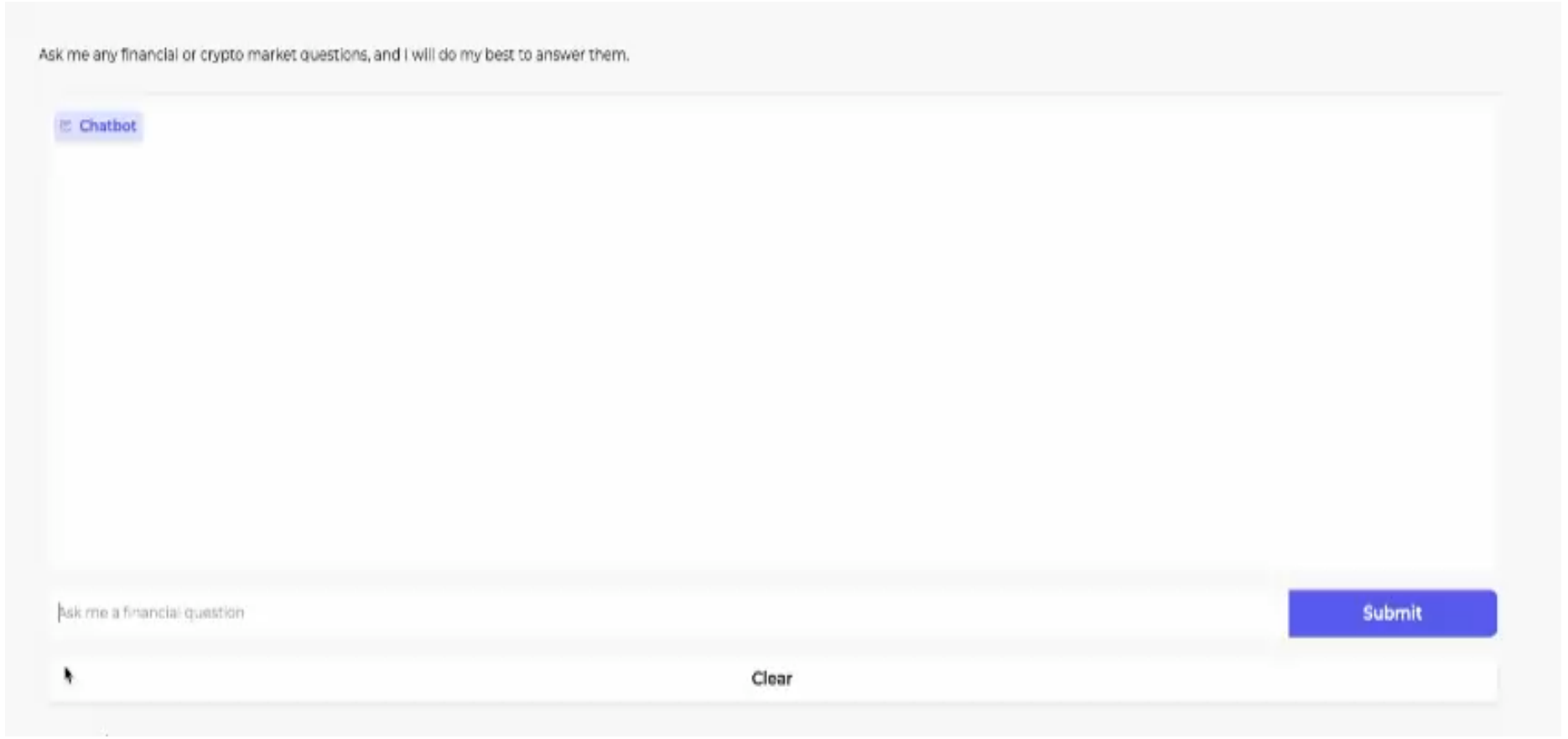
Improvement in Faithfulness:

• Hierarchical Indices: Ensures the retrieved data is contextually correct, reducing hallucinations by prioritizing high-quality sources.

• Summarization: Retains the essence of the source material, minimizing errors during response generation by focusing on core facts.



Base line example



Indexify.AI example

Planned Improvements

Enhancing Hierarchical Storing

Using a better hierarchically-organized database to improve inference time.

Batch API Calls

Batching the GPT API calls during streaming and inference to make them faster.

Better Hyperparameters and Queries

Further research to improve the top_k hyperparameter choices and enhance the queries, resulting in better results.

Thanks

Indexify.AI Team