QS STARS RATED FOR EXCELLENCE 2012

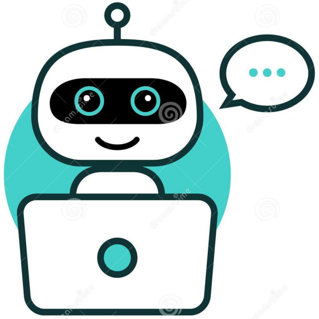
FPT Education

FPT UNIVERSITY

Developing a Chatbot Using Machine Learning:

A Case Study for US Stock Market

AIP490_G15



AIP490-G15

US_STOCK_TEAM		
Group Member	Nguyen Thanh Dat	HE151345
	Nguyen Ngoc Toan	HE151313
	Nguyen Thai Bao	HE151059
Supervisor	MSE. Le Dinh Huynh	

Table of Contents

01 Introduction

02 System Architecture and Requirements

03 System Design

04 Implementation

05 Case Study and Discussion

06 Conclusion

I. Introduction

1.1 Problem

We will build an AI Chatbot system. We will learn about chatbot, the applications and benefits of chatbots in businesses.

Customer care improvement

User experiment improvement

Purchase process simplification

Service integration opportunity

Resource saving

Personalized service

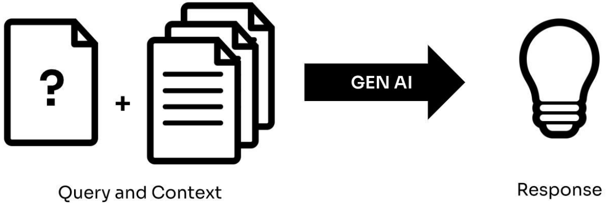
1.2 Related Work

We searched for ways to build a chatbot system that could respond using our data. There are several methods of using chatbot creation technologies without using code.

RASA

botpress

zapier

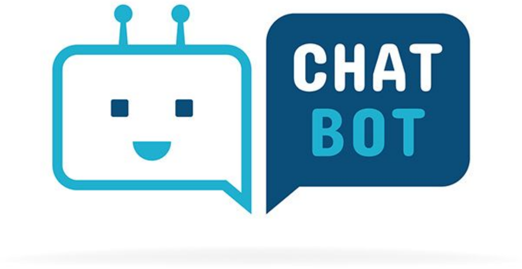


1.3 Objectives and Contribution

1. Engage in continuous learning about US stock technologies and sectors.

2. Build a chatbot for a website in the US stock domain.

3. Assist people in exploring our team's chatbot as a reference for building chatbots across various similar websites.

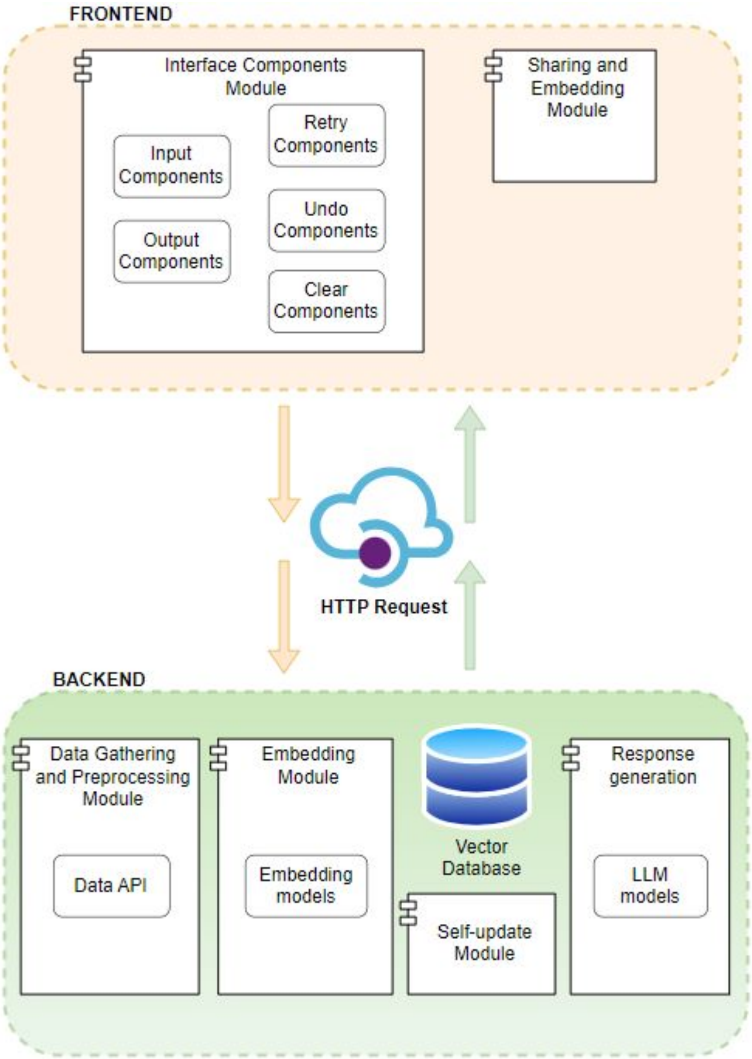


II. System Architecture and Requirement

2.1. System Architecture

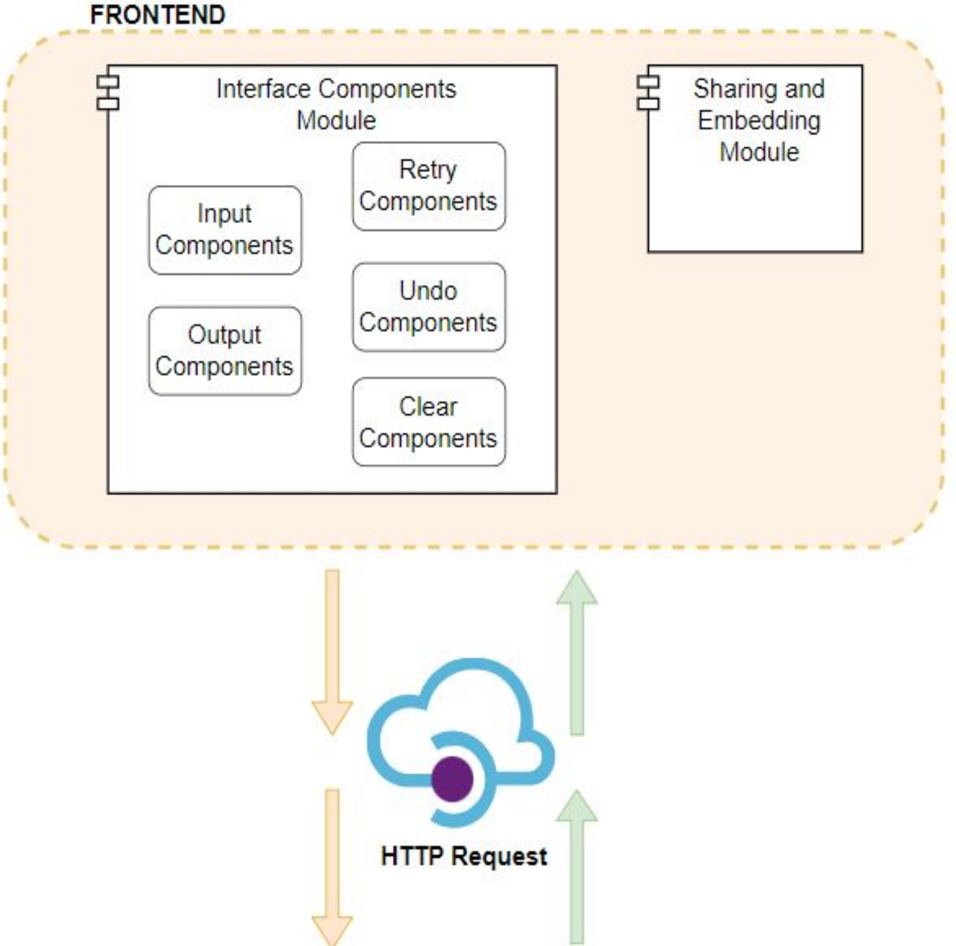
Front-end: Users can ask for information related to the field they are looking for.

Back-end: chatbot system processes these queries, delivering responses displayed on the same website's front end.



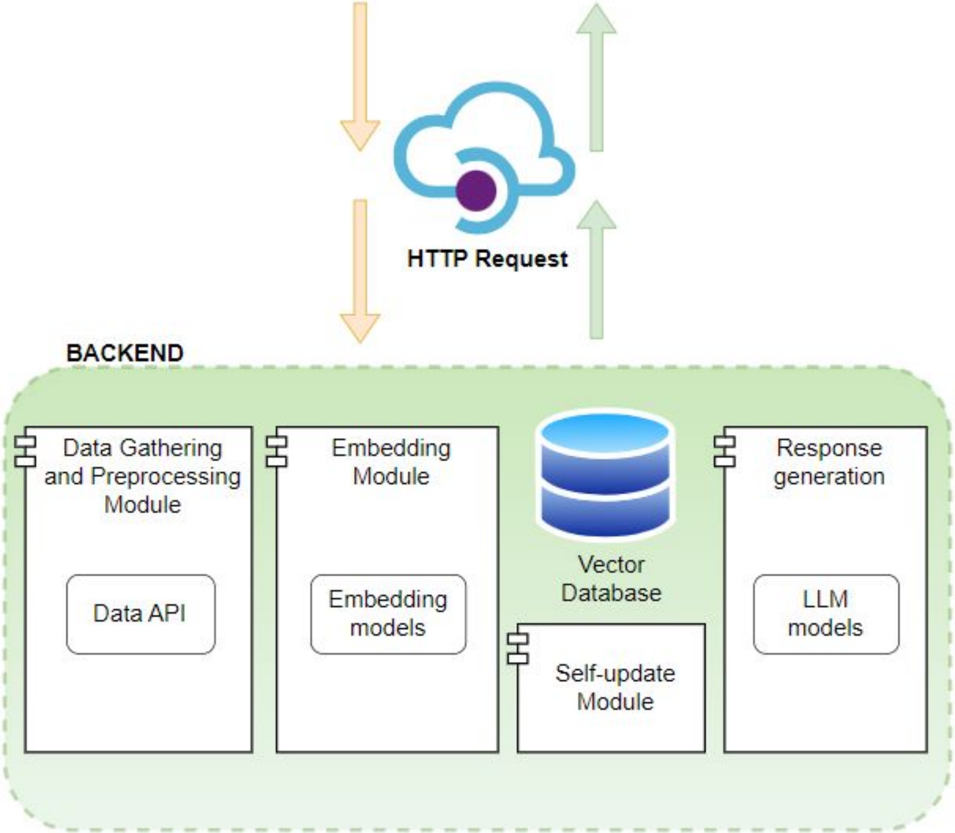
FRONT END

- Interface Components Module: includes input component which is the user part, output component, retry components, undo components, clear components.
- Sharing and Embedding Module: Share or embed directly on your website.



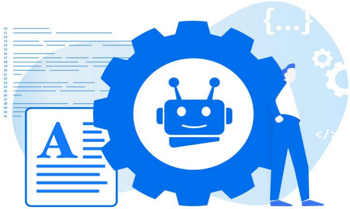
BACK END

- Data Gathering and Preprocessing module
- Embedding module
- Vectors Database module
- Self-update module
- Response generation module



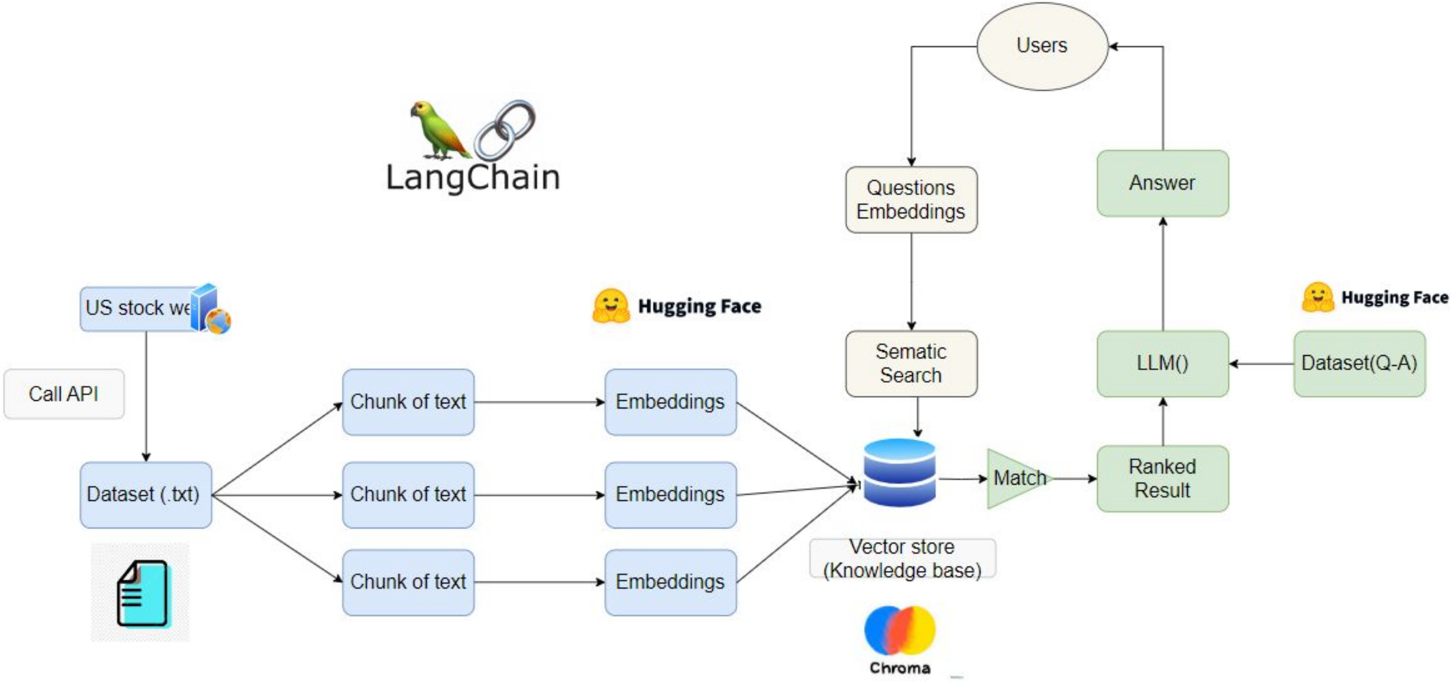
2.2. System Requirement

Group permissions		Content
Sysadmin	R.1	Dataset Management: update, add and fix
	R.2	System Configurations Management: create, update, and version control
Users	R.3	Customize by Sysadmin



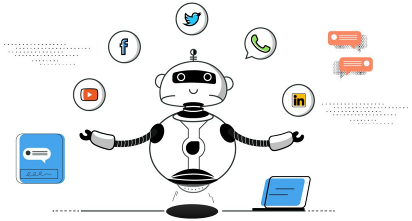
III. System Design

3. System Design



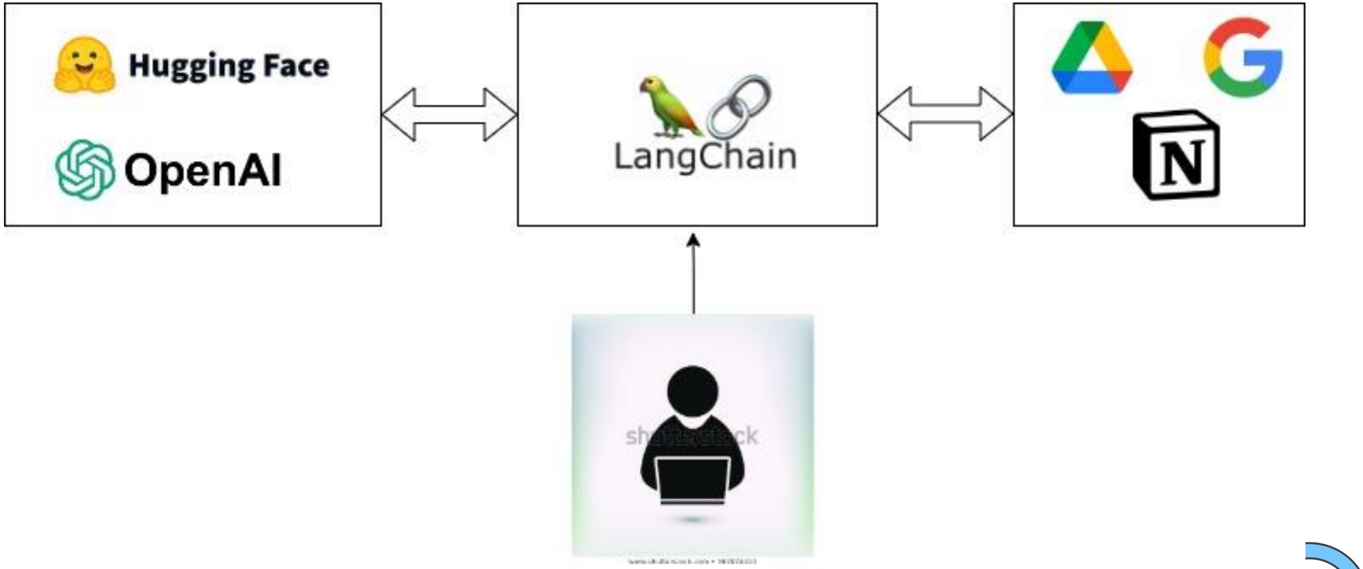
3. System Design

- Orchestration Framework: LangChain
- Data Extraction and Segmentation: LangChain
- Data Gathering and Processing: Utilizing the US stock API and WebSocket
- Embedding Model: llmrails/ember-v1
- Self-update Module: ChromaDB
- Vector Databases: ChromaDB
- Large Language Model (LLM): TheBloke/Llama-2-13b-Chat-GPTQ
- UX/UI : Gradio



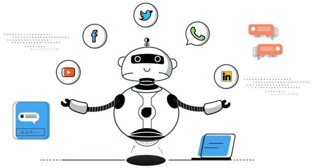
3. System Design

Langchain is a framework that helps you not only interact with major language models but also allows your application to take advantage of additional information from many other 3rd party data sources such as Google, Notion, Facebook.



3. System Design

Chroma DB is an open source, AI-native embedded database that aims to simplify the process of creating LLM applications by making knowledge, facts and skills connectable to LLM – as well as to avoid illusion.



3. System Design

	GPT - 4	**LLAMA-2**
Types of data (text, sound, images, etc.)	GPT-4 can handle more types of data	LLAMA-2 can handle less types of data
Data security	Low security	High security
Cost	Expensive	Free
Save resources	Low	High
Faster	Slow	Fast

3. System Design

As mentioned above, collecting financials data will use Restful API.

Advantages	Disadvantage
Easy to Understand and Implement	Limited Scalability
Easy Interoperability	Interactive Mode is Slower than gRPC
Easy Integration	Security Depends on Proper Setup

3. System Design

Websocket is a TCP-based transport protocol used to establish and maintain a two-way connection between a client and server through a single connection.

Advantages	Disadvantage
Two-way communication	Lack of Support on Some Environments
High performance	Difficult Confidentiality
Send real-time data	Possibility of Certain Security Risks

3. System Design

Finally, the interface of the chatbot I used for this project is Gradio. Gradio UI is an open source library used to create intuitive and interactive user interfaces (UIs) for applications and machine learning models.

IV. Implementation

4. Implementation

- Operating System: Google Colab Pro
- Graphics Processing Units: at least 16GB
- Random Access Memory: greater than 40GB
- Programming Language: Python

4. Implementation

Install the necessary libraries for the project such as: Langchain framework, Chroma DB, Pytorch, Websocket, Gradio.

4. Implementation

We use Websocket, Restful API methods to collect the current price change per second of stocks and other data on US stock website. After collecting data, we will read the data as a txt file.

```
def load_documents():
    loader = DirectoryLoader('/content/Data', glob="*.txt", loader_cls=TextLoader)
    documents = loader.load()
    return documents
docs = load_documents()
```

4. Implementation

Dividing the files into text chunks is a crucial step, particularly for embeddings. When a user poses a question, the system seeks precise numerical information for the answer.

```
def split_text_into_chunks(documents):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=70,
    chunk_overlap=0)
    text_chunks = text_splitter.split_documents(documents)
    return text_chunks
text_chunks = split_text_into_chunks(docs)
```

4. Implementation

The subsequent step involves embedding each of the paragraphs using the model (llmrails/ember-v1). Embeddings play a crucial role in mapping any text to a low-dimensional dense vector.

```
def create_embeddings():
    embeddings = HuggingFaceInstructEmbeddings(
    model_name="llmrails/ember-v1", model_kwargs={"device": DEVICE} )
    return embeddings
def create_vector_store(text_chunks, embeddings):
     db = Chroma.from_documents(text_chunks, embeddings,
     persist_directory="db")
     return db
db = create_vector_store(text_chunks, embeddings)
```

4. Implementation

This enables us to keep the base knowledge up-to-date, ensuring the chatbot can provide accurate responses to inquiries about the ever-changing stock prices.

```
while(True):
    loader = DirectoryLoader('/content/drive/MyDrive/data', glob="*.txt",
    loader_cls=TextLoader)
    documents1 = loader.load()
    text_chunks1 = split_text_into_chunks(documents1)
    list_id=db1.get(offset=906227,limit=50)['ids']
    if list_id:
      for i in range(len(text_chunks1)):
          db1.update_document(document_id=list_id[i],
document=text_chunks1[i])
```

4. Implementation

In the LLM part we use TheBlocke/Llama-2-13-B-chat-GPTQ to be able to generate answers word to users.

```
def create_llms_model():
    model_name_or_path = "TheBloke/Llama-2-13b-Chat-GPTQ"
    tokenizer = AutoTokenizer.from_pretrained(model_name_or_path,
    use_fast=True)
    model = AutoGPTQForCausalLM.from_quantized( model_name_or_path,
    revision="gptq-4bit-32g-actorder_True",
    use_safetensors=True,
    trust_remote_code=True,
    inject_fused_attention=False,
    device=DEVICE,
    quantize_config=None, )
    return model, tokenizer
```

4. Implementation

Next we will format the model's output using the prompt. The prompt will help guide the model to the required and appropriate output format for the user.

```
SYSTEM_PROMPT = "Use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't know, don't try to make up an answer."
template = generate_prompt(
    """
{context}
Question: {question}
""",
    system_prompt=SYSTEM_PROMPT, )
```

4. Implementation

Next we convert the text into numbers and configure text generation using the NLP model. Generate text with custom parameters such as creativity level, token count limit, and other parameters to control the text generation process.

```
text_pipeline = pipeline(
 "Text-generation",
  model=model,
  tokenizer=tokenizer,
  max_new_tokens=3096,
  temperature=0,
  top_p=0.95,
  repetition_penalty=0.9,
  streamer=streamer, )
llm = HuggingFacePipeline(pipeline=text_pipeline,
model_kwargs={"temperature": 0})
```

4. Implementation

Next, we build a process for retrieving information and answering questions based on the document content.

```
qa_chain = RetrievalQA.from_chain_type(
  llm=llm,
  chain_type="stuff",
  retriever=db.as_retriever(search_kwargs={"k": 2}),
  return_source_documents=True,
  chain_type_kwargs={"prompt": prompt}, )
```

4. Implementation

We used Gradio to build a user interface for a chatbot, allowing users to ask questions and receive answers from the chatbot through an intuitive interface.

```
import gradio as gr
def predict(message, history):
return qa_chain1(message)['result']
demo = gr.ChatInterface(
    fn=predict,
    title = 'ChatBot US_Stock StockScan.io'
    )
demo.launch(share=True)
```

4. Implementation

Finally, in the part of integrating chatbot software with the website, we use gradio as interfaces, Gradio provides features for sharing your machine learning model interfaces with others so there are two ways to integrate gradio on your website.

```
<iframe src="https://your-gradio-url" width="500" height="500"></iframe>
```

V. Case Study And Discussion

5.1 Stockscan.io US stock chatbot case

Line Century is a company related to US stocks, they have a website Stockscan.io which is a website about the stock market in the US.

StockScan.io	Function
Watch list	Table list includes (stock name, company name, price, %1D, volume, market cap..) over time (need to buy web package)
Top list	List of top 5 stocks that increase or decrease over time (today, 1 week, 1 month, 6 month), list of top 5 penny stocks, OTC
Option	Table list includes (stock code name, Call/Put order, Strike, Price, Change, %Change, Volume)
Financial	Financial Data by each company (chart format) including (revenue, net income, Cash flow, EPS, Debt to Equity Ratio)
Price History	Historical price list by time (daily, weekly, monthly), by year, table includes (Date, high, low, high-low, volume, %change)

5.1 Stockscan.io US stock chatbot case

Data parameters	**Define**
Stock	The capital raised by a business or corporation through the issue and subscription of shares.
Current Price	Current price of stock.
Revenue	Revenue is the result of regular business activities, computed by multiplying the average sales price by the quantity of units sold.
Net income	Net income (NI) is determined by subtracting expenses, interest, and taxes from revenues.
Cash flow	The cash or cash-equivalent that a company receives or disburses as payments to creditors.
EPS	Earnings per share (EPS) is computed by dividing a company's profit by the total number of outstanding shares of its common stock.
D/E	The Debt-to-Equity (D/E) ratio evaluates a company's total liabilities in relation to its shareholder equity, providing insights into the degree of reliance on debt.

5.1 Stockscan.io US stock chatbot case

In addition, we can use the API Restful to collect the data we need to collect data in website US stock: financial data(Revenue, Net Income, Cash Flow, EPS, D/E).

```
def financial_data(url, exchange_slug, symbol):
  data = { 'exchange_slug': exchange_slug,
        'symbol': symbol }
  response = requests.post(url, json=data)
  if response.status_code == 200:
    try:
      result = response.json()
      return result
    except json.JSONDecodeError as e:
      return None else:
urls = [ “Your URL”]
exchange_slugs = ["NASDAQ"] # Add more exchanges if needed
symbols = ["AAPL"] # Add more symbols if needed
result_filenames = { urls[0]: "Revenue.txt",
                urls[1]: "Net-Income.txt",
                urls[2]: "Cash-Flow.txt",
                urls[3]: "EPS.txt",
                urls[4]: "DTER.txt" }
financials_labels = ["Revenue", "Net Income", "Cash Flow", "EPS", "DTER"]
```

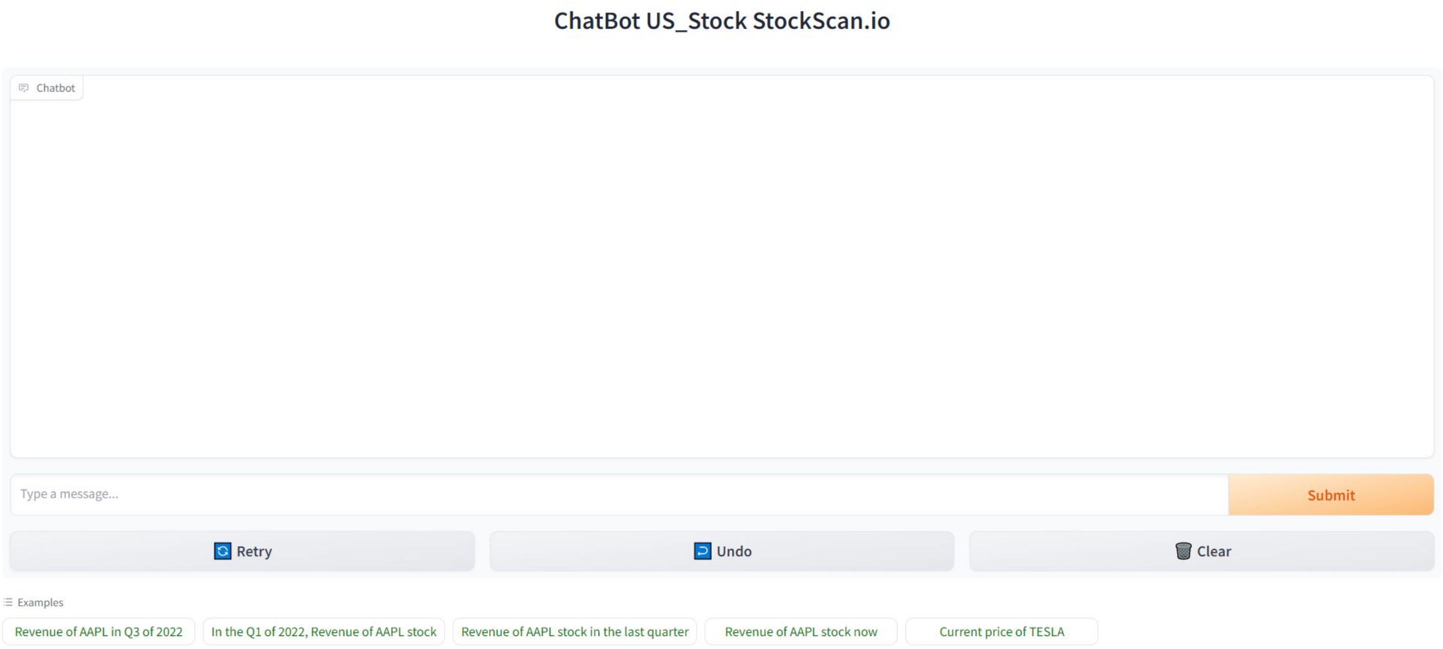
5.1 Stockscan.io US stock chatbot case

In the price part of the stock code, we use websocket to collect stock prices, and it will be continuously updated so that the chatbot can give reasonable results.

```
sio = socketio.Client(logger=True, engineio_logger=True)
@sio.on('connect')
def on_connect():
    sio.emit("RealTimeAvgPriceSubAdd", {
        'subs': listcoins
    })
@sio.on('avg_price_update')
def handle_global_price_update(data):
                if all(symbol in coin_prices for symbol in listcoins):
               with open('/content/drive/MyDrive/data/prices.txt', 'w') as txt_file:
                 for symbol, price in coin_prices.items():
                   txt_file.write(f"Current price of {symbol} stock is {price}$\n")
@sio.on('disconnect')
def disconnect():
    print('Disconnected')
sio.connect(url='Your WSS URL', transports=['websocket'])
```

5.1 Stockscan.io US stock chatbot case

Finally, we apply section Implementation above to build an AI chatbot for the website stockscan.io.



ChatBot US_Stock StockScan.io

Revenue of AAPL in Q3 of 2022

In the Q1 of 2022, Revenue of AAPL stock

Revenue of AAPL stock in the last quarter

Revenue of AAPL stock now

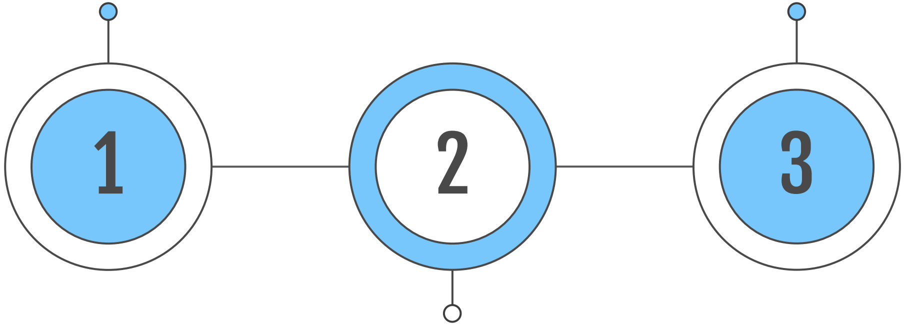
Current price of TESLA

5.2 Discussion

This AI chatbot is being used to answer 2000 US stock tickers and has the performance of being able to answer each question within a period of 3-5s.

	Execution Time (m)
Data gathering	60m
Training LLM(TheBloke/Llama-2-13b-Chat-GPTQ)	5m
Embedding and Save vector to chroma db	120m
Update vector 50 stock	0.05m - 0.08m → 3s -5s
Answer question	0.05m - 0.08m → 3s -5s

VI. Conclusion



Engage in continuous learning about US stock technologies and sectors.

Build a chatbot for the stockscan.io website.

Develop a chatbot to provide assistance and guidance to users.

Thank for listening!

Do you have any questions?

nguyenngoctoan2001bn@gmail.com
nguyenthanhdat020501@gmail.com
bao24901qaz@gmail.com