QS STARS RATED FOR EXCELLENCE 2012

FPT Education

FPT UNIVERSITY

Graduation Thesis Final Report

Developing a Chatbot Using Machine Learning: A Case Study for US Stock Market

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

A thesis submitted in partial fulfillment of the degree of BSc.(Hons.) in Artificial Intelligence with the supervisor of MSE. Le Dinh Huynh

Bachelor of Computer Science
Hoa Lac Campus - FPT University
2023

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to our research mentor, Mr. Huynh, for affording us the invaluable opportunity to engage in research and for providing unwavering guidance throughout this endeavor. His dynamism, visionary insights, sincerity and motivation have been a profound source of inspiration for us. Under his tutelage, we learned the methodologies essential for conducting research and presenting our work clearly. In addition to our mentor, we would like to express our gratitude to our friends at FPT University for their steadfast support, patience and camaraderie. Our sincere appreciation also extends to FPT University for fostering an exceptional environment that contributed significantly to our development over the four years of our studies. Last, but certainly not least, we want to acknowledge the unwavering support of our caring and loving families. Their encouragement provided great comfort and solace during challenging times.

ABSTRACT

Instead of conventional searches on websites, there is a trend of creating chatbots to ask and answer questions with customers, increasing interaction and customer experience. In this work, we present a chatbot system in the financial sector. Chatbots working with customers handle the website platform, customers can note the development of an exciting, great quality platform leading to more conversions and outstanding leads. This is especially true for businesses that need to attract many customers. Chatbots can provide instant information and answer users questions quickly, helping them access information effectively. Chatbots can create intuitive and enjoyable interactive experiences that help attract and maintain customer interest. This article presents the collaborative design between the Langchain framework and large language models (LLMs) such as ChatGPT, LLaMA....Large language models have demonstrated great potential in natural language processing tasks in the financial sector. Langchain framework is deployed on many projects, helping to significantly reduce time and costs. The chatbot system helps people access information faster. Normally, searching for financial information takes about 7-10 seconds, while the chatbot system extracting information takes about 3-5 seconds. Stockscan.io is a website in this field. In the financial sector, it is also a test case to test the system. The chatbot system can be applied to fields beyond finance and can be further improved.

Keyword: ChatBot AI, Large Language Model, RAG, Langchain, US stock

CONTENTS

I. INTRODUCTION

1.1 Problem

In this article, we will build an AI Chatbot system for websites about US stocks. First of all, we will learn about the applications and benefits of chatbots in businesses described in ***Fig.1***. Chatbots offer businesses numerous applications and advantages. They enhance customer experience by providing instant support and guidance, optimizing customer support processes and reducing operational costs. Additionally, chatbots serve as effective marketing tools, engaging customers and promoting products. Their interactions also yield valuable insights into customer behavior, enabling businesses to refine strategies and make informed decisions. Automation, a vital need for every business, is addressed as chatbots contribute to automating customer care work, particularly crucial in the fast-paced and dynamic stock market environment. The growing necessity for chatbots is emphasized, especially considering the limited availability of specialized systems for the stock market. Hence, owning a chatbot system becomes a significant advantage, enhancing the business's market position and attracting more users to the business. In essence, integrating chatbots into business strategies improves competitiveness, streamlines operations and adapts to the dynamic demands of the modern business landscape.

Website stock is a website that provides information about the stock market, stock prices, market indices, economic news and other financial services. Stock websites often provide tools for investors and traders to track and analyze market data, making smart investment decisions. Building a chatbot website offers several significant advantages. Firstly, it enhances customer experience. Secondly, it saves time as chatbots can handle multiple requests simultaneously, operating 24/7 to improve service quality. Thirdly, incorporating Chatbot AI adds an interesting feature, increasing the attractiveness of your website and attracting more customers by delivering valuable stock market information.

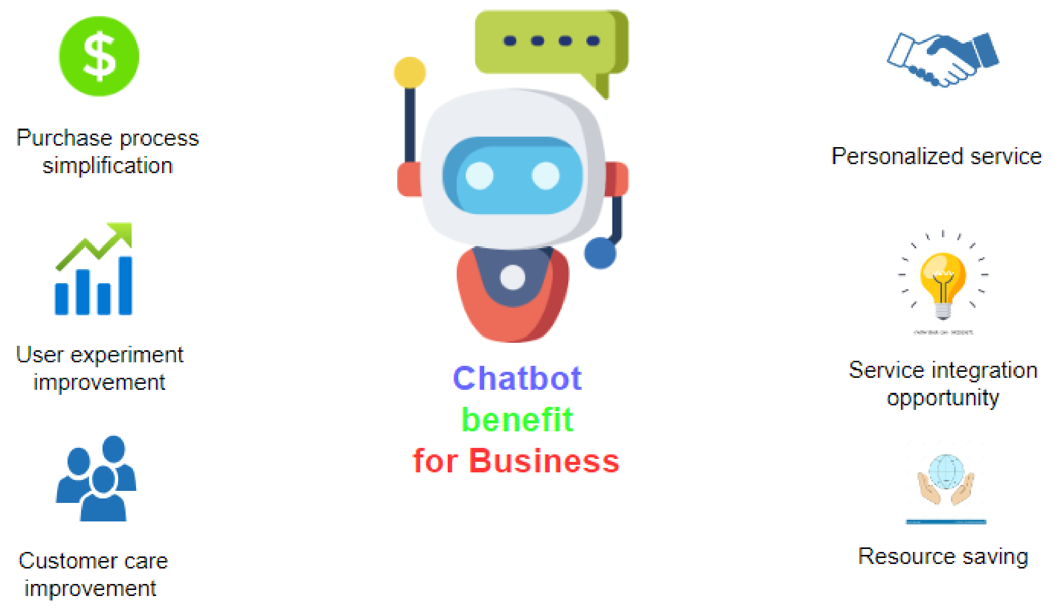

Purchase process simplification
User experiment improvement
Customer care improvement
Chatbot benefit for Business
Personalized service
Service integration opportunity
Resource saving

Fig. 1. Benefits of chatbot for business

1.2 Related work

Cheonsu Jeong technical leader for ai automation platform at Samsung [1] using an enterprise data-based LLM application architecture using large language models, a retrieval-augmented generation (RAG) algorithms models, OpenAI's GPT, Google's Bert, AI chatbots have great power in answering questions using internal data. Myeong-Ha Hwang. et al [2] using Rasa, NLU natural language processing tool, this AI chatbot includes 1506 sentences, about 51 intents daily conversion and has an efficiency of 0.82. Yeon Seonwoo. et al [3] using SQuID-BM25/DPR, SQuID-RePAQ/DPR natural language processing tool. The performance of BM25 and RePAQ are 64.07% and 64.34% on NQ and 61.73% and 59.10% on TriviaQA. Department of computer science and engineering school of computing Sathyabama [4] built a full stack chatbot system with Google API, the system includes back end and front end, in the back end there will be a system chatbot that can search from the database and respond to data from Google API.

Isabelle Augenstein. et al [5] using large language models (LLMs), such as OpenAI's ChatGPT, Microsoft's Bing Chat, and Google's Bard, Face to Face with the largest language models that are highly effective in answering questions that have many challenges to face. Deussom Djomadji Eric Michel college of technology university of buea department of electrical and electronic engineering. et al [6] using integrated natural language processing (NLP) technology, creating a conversational agent for the supervision of security events using SIEM tools. Ahmad Abdellatif. et al [7] using NLU machine-learning algorithms, Rasa, integrating natural language processing (NLP) technology, Med. TimeToAnswer of the chatbot is 14.8s and has an accuracy of 0.7. Adith Sreeram A S. et al [8] using LangChain and Large Language Model, chatbot uses data from PDF with strong conversation answerability.

Hana Demma Wube. et al [9] In the study conducted, 6% of the participants refrained from utilizing chatbots, citing reasons such as 40% lacking access to the necessary technology. Additionally, the research revealed that 76% of the interviewees expressed dissatisfaction with the bank chatbot technology, while 24% reported satisfaction. Zhiyu Chen. et al [10] the utilization of RoBERTa in studying and analyzing the performance of current pre-trained models in complex and specialized domains was undertaken. With a BERT-based retriever, the study achieved an 89.66% recall for the top 3 retrieved facts and a 93.63% recall for the top 5. In comparison, using TF-IDF resulted in an 82.91% recall for the top 5 facts. Qianqian Xie. et al [11] leveraging large language models (LLMs), we demonstrated the efficacy of FinMA across diverse financial tasks. This highlights the potential of domain-specific instruction tuning for large language models within the financial domain. The results indicate promising performance, with FinMA 7B achieving 0.86%, FinMA 30B reaching 0.87% and FinMA 7B-full also achieving 0.87%. Rudi Setiawan. et al [12] instruction fine-tuning yields notable performance enhancements, with improvements of 4.6%, 24.0% and 9.0% observed in MTEB, billboard and prompt retrieval, respectively. Notably, in various task categories, Instructor-Large (335M) demonstrates substantial improvements over GTR-Large, particularly in text evaluation (24.0%), prompt retrieval (9.0%) and classification tasks (7.3%).

James C. L. Chow. et al [13] leveraging LLMs and IBM Watson, we have developed a real-time bot dedicated to disseminating healthcare knowledge to the public. To further enhance the chatbot's performance and provide an improved conversational experience, several improvements have been proposed. These include the incorporation of additional features like language translation or text-to-speech conversion, exploration of alternative conversational approaches for more effective knowledge transfer and integration of ChatGPT into the system. These enhancements aim to offer a more versatile and user-friendly healthcare knowledge dissemination platform. Shamane Siriwardhana. et al [14] introduced a novel extension of RAG called RAG-end2end and conducted an evaluation using three datasets from diverse domains (COVID-19, News and Conversations). The results demonstrate that RAG-end2end achieves substantial performance improvements across all three domains compared to the original RAG implementation. Shuo Li. et al [15] we introduced a novel strategy that applies conformal prediction to retrieval-augmented question answering. Additionally, we employed Bayesian optimization to effectively select hyperparameters for the global test, aiming to maximize the overall performance of the system, resulting in an impressive coefficient of determination (Cov) of 0.92. Nura Esfandiari. et al [16] we introduced a novel model that combines cWGAN and transformer architecture. The outcomes substantiated the superiority of our proposed model over state-of-the-art approaches, as evidenced by significant improvements in bleu4, rouge-l, f-measure and meteor metrics. Leveraging the capabilities of cWGAN and the transformer model, the proposed model excels in generating precise, semantically relevant and human-like answers.

Jiangtong Li. et al [17] Developing CFGPT for a Chinese financial assistant involves utilizing a large language model, CFDataQA, comprising 12 thousand instances and 6 million tokens specifically designed for question-answering. The effectiveness of our CFLLM-ins-7B model is demonstrated across various financial tasks, highlighting the significant potential of domain-specific continued pretraining and supervised fine-tuning of large language models within the financial domain. Hugo Touvron. et al [18] the article comprehensively covers all aspects of Llama-2, providing a thorough understanding of its key elements. It delves into topics such as the definition of Llama-2, its model architecture, the training process encompassing pre-training and supervised tuning, as well as aspects related to model safety. This inclusive coverage ensures readers gain a holistic insight into the Llama-2 model and its various components. Kim Martineau [19] wrote this article on IBM Research's blog that gives us an overview of the RAG method and its functions, thereby drawing out the advantages and disadvantages of this method. Jiawei Chen. et al [20] the assessment focused on four key capabilities of retrieval augmented generation in LLMs: noise robustness, negative rejection, information integration and counterfactual robustness.

1.3 Objective and Contribution

In this project, our primary objectives include gaining knowledge about chatbots and associated technologies, as well as acquiring practical experience in building chatbots. Our secondary goal is to construct a specialized chatbot tailored for the US stock market, with a specific focus on the website stockscan.io. The overarching and desired aim of our team is to create a chatbot architecture that can serve as a reference for others, facilitating the implementation of chatbots on similar websites. The flexibility of our architecture allows it to be adapted to any website and we have chosen stockscan.io as a test case. To enhance the quality of responses, we employ text generation models, ensuring not only richness in answers but also maintaining data accuracy. This multifaceted approach aligns with our vision of creating a versatile and effective chatbot solution.

II. SYSTEM ARCHITECTURE AND REQUIREMENT

2.1 Chatbot system architecture overview

The architecture of the chatbot system is bifurcated into two primary components: the user interface (front-end) and the back-end (refer to ***Fig. 2***). Users can inquire about information regarding the US stock sector on the website's front-end and the back-end chatbot system processes these queries, delivering responses displayed on the same website's front-end. Various frameworks like Gradio, Streamlit, Chanlit, etc., can be employed in the front-end to construct user interfaces for machine learning and computer vision models. Alternatively, a custom front-end can be developed using popular front-end frameworks such as Node.js and JavaScript. The front-end comprises essential components like input boxes, output boxes, submit buttons, clear buttons, undo buttons and retry buttons. The back-end encompasses a Retrieval-Augmented Generation (RAG) module that handles data gathering, preprocesses data, embeds data, manages databases and includes a self-update module alongside the Large Language Model (LLM) module. The front-end is equipped with built-in modules that define the display layout.

- *Interface components module* includes input component which is the user part, output component, retry components, undo components, clear components.

- *Sharing and Embedding module is* share or embed directly on your website.

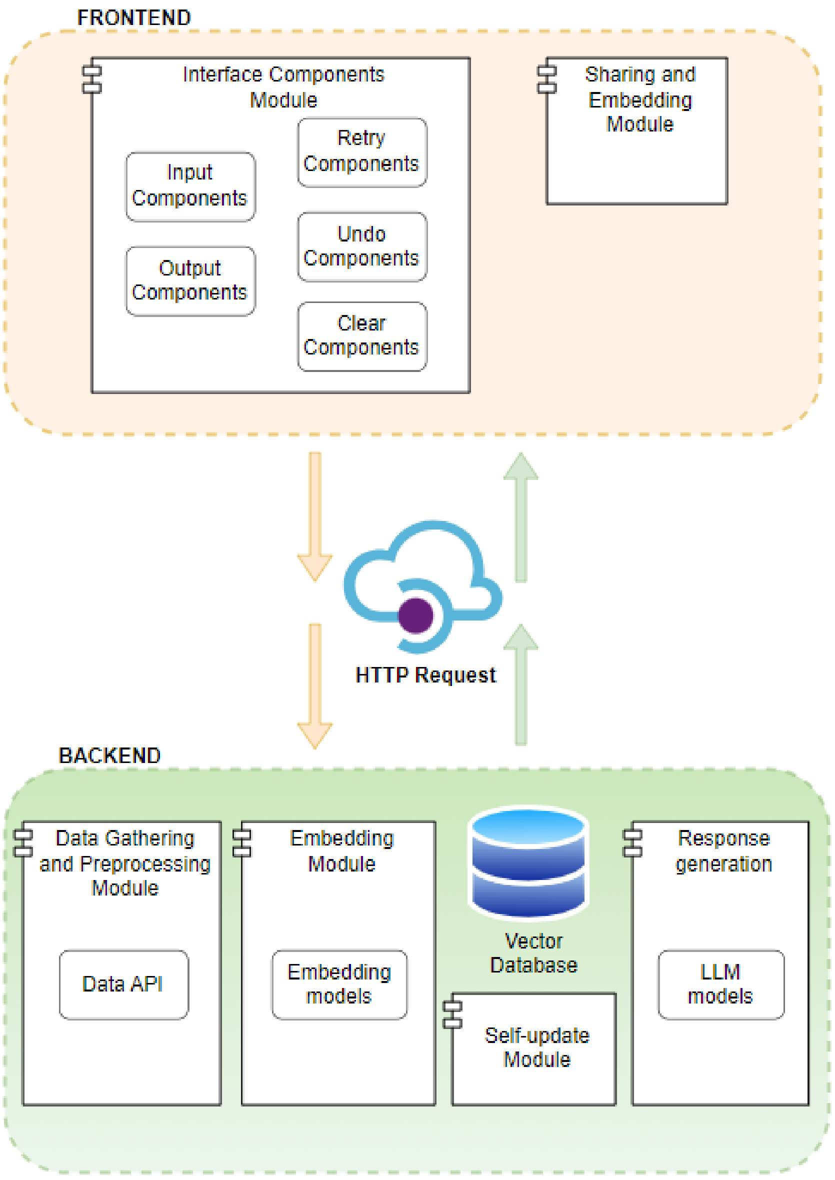


Fig. 2. General chatbot architecture

The backend handles all data operations and technical.

- *Data Gathering and Preprocessing* module takes care of collecting data from the website and processing it so that it can be included in the embedded module in various ways such as US stock API or Websocket from the website.
- *Embedding module* transforms textual data into vectors, utilizing embedding models from platforms such as Hugging Face or GPT.

- *Vectors Database module* is a form of NoSQL storage that stores data in multi-dimensional vector form, employing solutions like Chroma DB, Faiss and others. It can be implemented as a local storage solution or integrated with cloud storage platforms such as Pinecone.
- *Self-update module* takes care of updating stock prices that change over time into the database.
- *Response generation* module to answer user questions in context based on database data including paid model APIs such as GPT 3.5, GPT4 or open source LLMs on hugging face such as: llama2, zephyr.

2.2 System Requirements

The system has two main group permissions. Group 1 will be Sysadmin and group 2 will be Users. Sysadmin with the right to manage the database containing data and the right to edit information so that the chatbot can answer according to the admin's wishes. Users have the right to manipulate chatbots to find information they need to know. Below is ***Table 1*** giving a summary of the system's functions.

Table 1. Functional requirements

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

Section 2 described numerous required functional requirements and elements in UI. Almost frontend and backend modules are standardized. The proposed tool needs to use design analysis for integration and interoperability. The next section will discuss this issue.

III. SYSTEM DESIGN AND IMPLEMENTATION

3.1 System design

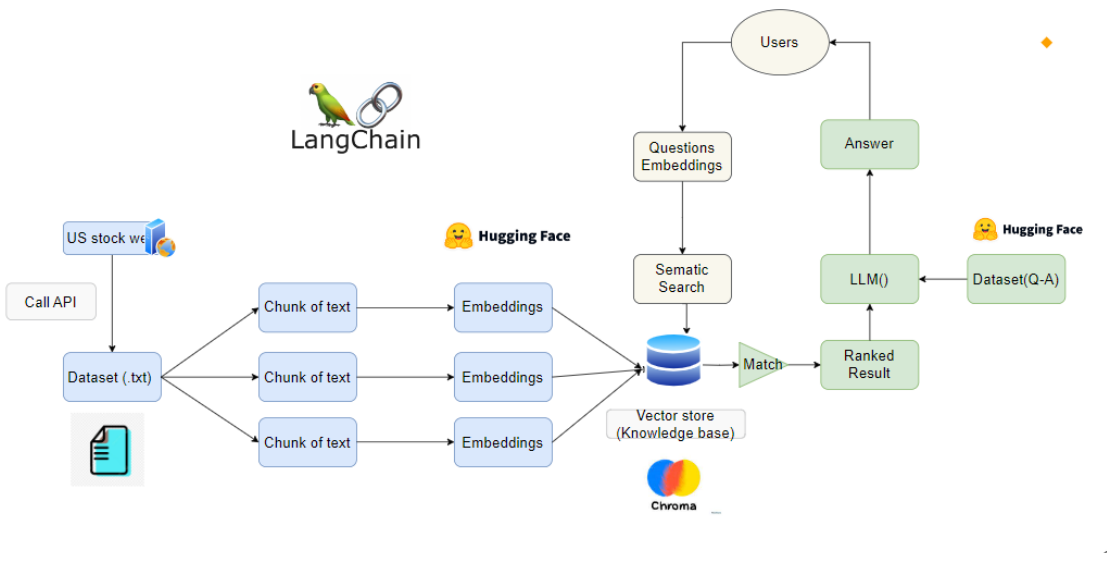


Fig. 3. RAG module chatbot AI system

The primary challenge addressed and a pivotal aspect of this project revolves around the utilization of the Retrieval-Augmented Generation (RAG) method (refer Fig .3). RAG is a method introduced by researchers from Meta AI. RAG is a combination of the information retrieval component with text generation models. This approach plays a crucial role in enhancing the performance of AI chatbots, specifically those based on Large Language Models (LLMs). RAG involves integrating external sources of knowledge to augment the internal representation of information within the LLM generated responses. Implementing RAG within an LLM-based question-answering system yields two key advantages. Firstly, it ensures the model's access to the latest and most reliable facts. Secondly, it provides users with transparency by enabling them to access the model's sources. This transparency allows users to verify the accuracy of the model's claims, instilling trust in the system.

To provide a comprehensive overview of the advantages and disadvantages of chatbots employing the RAG method, we will present a table delineating these aspects. Besides the advantages, there will be disadvantages of the RAG method as shown in ***Table 2.*** The simple structure of the RAG method will be represented as The questions raised and processed by LLM will then be searched for answers to that question based on our base knowledge.

Table 2. Advantages and disadvantages of RAG

Advantage	Disadvantage
High level of information security.	Crafting responses for every conceivable question a customer might ask required a significant amount of time.
This diminishes the likelihood of an LLM inadvertently disclosing sensitive data or generating inaccurate or misleading information through 'hallucination.'	Updating streams is difficult and time consuming.
Implementing RAG has the potential to reduce the computational and financial burdens associated with operating LLM-powered chatbots within an enterprise environment.	In the absence of accounting for a particular scenario, the chatbot lacked the capability to improvise.

Based on ***Fig. 3.*** This comprehensive set of system design combines the LangChain framework for orchestration, LangChain modules for data extraction and segmentation, the US stock API and WebSocket for data gathering and processing, llmrails/ember-v1 for embedding models, ChromaDB for self-update and vector databases, TheBloke/Llama-2-13b-Chat-GPTQ for a powerful language model and Gradio for the user experience/user interface (UX/UI). Together, these components create a sophisticated AI system designed to meet diverse and complex needs.

- 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

Our project's orchestration framework uses the Langchain framework. Langchain is an extremely hot framework in recent times. It was created to harness the power of large language models (LLM) such as ChatGPT, LLaMA, etc., to develop real-world applications. Despite being developed just over a year ago (since October 2022) and continuously updated daily, Langchain has received tremendous interactions on GitHub, boasting more than 70K stars. This is equivalent to the star count of another legendary framework in deep learning, PyTorch, which achieved this recognition after more than six years of dedicated development and effort. Langchain is a framework that allows your application to take advantage of additional information from many other 3rd party data sources such as Google, Notion, Facebook... as well as providing components that allow the use of language models in many different real-life situations. We can imagine Langchain as a bridge between the language model and third-party applications as follows in Fig. 4.

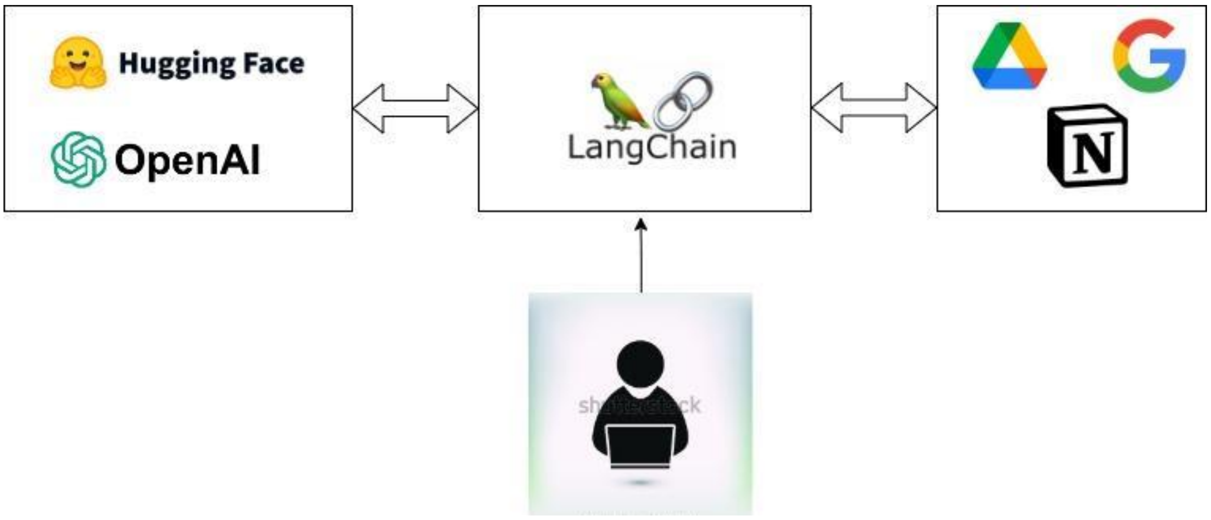


Fig .4. Langchain Framework Flow

There are two main advantages of the langchain framework:

- Provides diverse components: LangChain provides a variety of components necessary for interacting with language models. These components are designed to be easy to use, extend and customize for many different problems.

- Providing chains for specific use-cases: A chain is understood as a series of components paired together in a certain order so that real-life use cases can be solved. The use-cases that langchain provides are virtual assistants, document-based Q&A, chatbots, support for querying table data, interaction with APIs, text feature extraction and text evaluation, text summary.

The langchain itself does not contain models but it will contain interfaces to help interact with the model more easily, Models can be saved on OpenAI's system, Hugging Face.... Prompt templates: Recently we have heard of the concept of prompt engineering, which means techniques for writing prompts for the language model so that it can correctly answer the problems we want the language model to perform. A prompt template is simply a piece of text that includes instructions, requirements and examples to include in the language model and it can receive input as parameters passed by the user.

Text Embedding Model: Simply put, embedding is a representation of a piece of text. It will be represented by a vector with a fixed number of dimensions, for example 128 dimensions. The closer two vectors are, the more similar they are in semantics or content. Langchain allows us to connect to different types of embedding models called Embedding providers (OpenAI, Cohere, Hugging Face, etc). And currently the strongest embeddings model in the MTEB leaderboard proposed by Hugging Face is the llmrails/ember-v1 model. So we have used it, you can completely change this model because the model will be improved over time. We highly recommend that you use the most powerful model so that your system can find the most accurate answer.

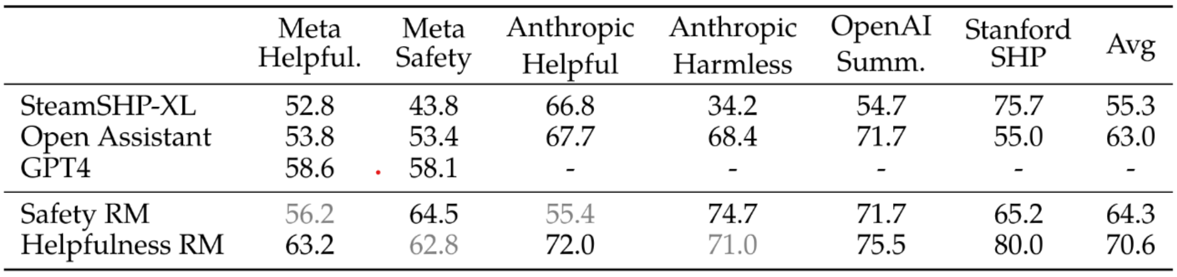
To have a base knowledge for the chatbot system, we use Chroma DB. Chroma DB is an open source, AI-native embedded database that aims to simplify the process of creating LLM applications by making knowledge, facts and skills connectable to LLM – as well as to avoid illusion. Chroma DB has many important features such as: querying, filtering, density estimation, update and more. In particular, the Langchain framework fully supports Chroma DB.

With LangChain as Our project's Orchestration framework, we can use many different LLM models, especially the most powerful LLM models today such as GPT 4, LLama-2, ... However, after considering ***Table 3*** we have chosen the LLM 'TheBloke/Llama-2-13b-Chat-GPTQ' that fits our case study as well as this chatbot system. This is the LLaMa-2 model that has been refined thanks to TheBloke and can be used for commercial purposes. In short, LLaMa-2 is the next version of LLaMa - a large language model created by Facebook AI Research and their team of engineers. This model is architecturally similar to LLaMa but adds data, improves quality and adds new optimization methods to achieve higher performance. Below is a comparison between LLAMA2 and GPT4 models in ***Table 3.***

Table 3. Comparison between GPT-4 and LLAMA-2

	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	Lower	Higher
Speed	Slower	Faster

The effectiveness of the final helpfulness and safety reward models is evaluated across a varied array of human preference benchmarks. Notably, our model undergoes fine-tuning on the collected data, distinguishing it from the other baselines presented in ***Fig. 5.***



	Meta Helpful.	Meta Safety	Anthropic Helpful	Anthropic Harmless	OpenAI Summ.	Stanford SHP	Avg
SteamSHP-XL	52.8	43.8	66.8	34.2	54.7	75.7	55.3
Open Assistant	53.8	53.4	67.7	68.4	71.7	55.0	63.0
GPT4	58.6	58.1	-	-	-	-	-
Safety RM	56.2	64.5	55.4	74.7	71.7	65.2	64.3
Helpfulness RM	63.2	62.8	72.0	71.0	75.5	80.0	70.6

Fig. 5. Reward model results(Hugo Touvron, Louis Martin, Kevin Stone.et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. 2023. pp12)

The process commences with the pretraining of Llama 2, leveraging publicly accessible online sources. Subsequent to this phase, we generate an initial iteration of Llama 2-Chat by employing supervised fine-tuning. The model then undergoes iterative refinement using Reinforcement Learning with Human

Feedback (RLHF) methodologies, specifically incorporating rejection sampling and Proximal Policy Optimization (PPO). It is imperative during the RLHF stage to accumulate iterative reward modeling data in tandem with enhancing the model to ensure the reward models remain within distribution, as illustrated in ***Fig. 6.***

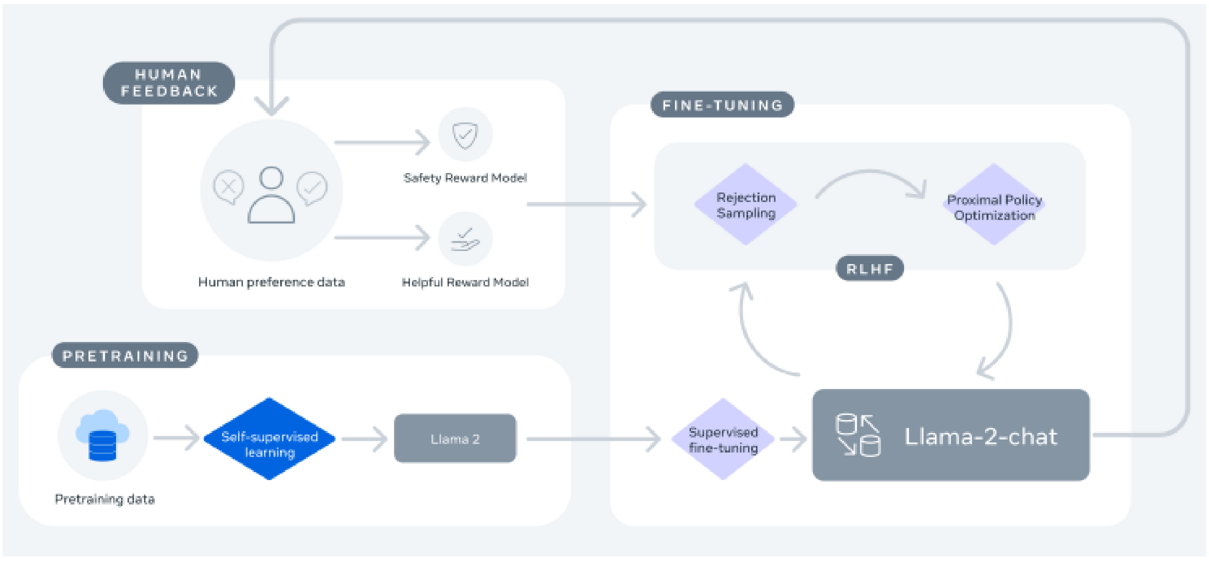


Fig .6. Training of Llama 2-Chat(Hugo Touvron.et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. 2023. pp5)

In terms of data collection for the chatbot system's base knowledge, we use websocket to collect stock prices and restful API for financial data. First let's get to websocket, it is a TCP-based transport protocol used to establish and maintain a two-way connection between a client and server through a single connection. It allows real-time data transmission and continuous interaction between client and server, without the need to establish a new connection each time new data needs to be sent or received.

The protocol has two parts: handshake and data transfer. Initially, the client sends a request to initiate a websocket connection to the server, the server checks and returns the results accepting the connection. Then the connection is created and the sending process data can be taken, the main data is the Ws frames. Therefore, we use this technique for this project, helping the chatbot to continuously update stock prices in the most accurate way. More specifically, we use the Socket.io library. ***Table 4*** that is about the advantages and disadvantages of Websocket.

Table 4. Advantages and Disadvantage of Websocket

Advantages	**Disadvantage**
Fast connection at fewer overheads	The functionality may not be supported if the browser does not fully comply with HTML5 standards.
Request/response streaming in real-time	It doesn't support edge caching
High performance	Possibility of Certain Security Risks

As mentioned above, collecting financials data will use Restful API. RESTful API (Representational State Transfer) is a software architecture designed to create web services based on REST principles. REST is an architectural style specifically designed for web-based distributed systems. It is distinguished by its emphasis on simplicity, scalability and seamless interoperability between applications. In RESTful APIs, everything considered a resource, such as objects or data, has a unique identifier (URI). For example, a resource can be a product, a user, or anything you want to perform an operation on. RESTful API uses HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources. For example, you can use the GET method to get information about a resource, POST to create a new resource, PUT to update a resource and DELETE to delete a resource.

Resource data, typically expressed as JSON or XML, about the current state of the resource. Resources can have many different states and can be changed via HTTP operations. The REST model is stateless, meaning that every request from the client must include all the information needed to serve that request. The server does not need to save the client's state between requests.

To build the chatbot system we used Google colab Pro to deploy the code. Google Colab Pro is the premium version of the Google Colaboratory service. It offers several benefits and expanded features compared to the free version of Google Colab. Here are some important points about Google Colab Pro:

In Google Colab Pro, you have more powerful GPU and TPU usage than in the free version. This is especially useful for machine learning and heavy number crunching tasks. In the free version, you have a limit on GPU usage time. However, with Colab Pro, you can use GPU and TPU continuously and without a time limit. If you do not use google colab pro type A100 GPU, you can use Ubuntu server with hardware including 30GB drive, 30GB GPU RAM, 30GB system RAM.

Finally, the interface of the chatbot we 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. Here are some important features of Gradio UI: Gradio UI is designed to be simple and easy to use. Gradio UI supports input data types such as text, images, audio, video and allows visual results to be displayed in a variety of formats.

Gradio UI can integrate with machine learning models built with TensorFlow, PyTorch, Scikit-learn, and other libraries. You can connect your machine learning model to Gradio UI in just a few lines of code. Gradio UI allows you to create realistic applications with support for webcams and microphones to make the user interface more interactive.

3.2 Implementation

Python is chosen as the programming language for its widespread use in AI development, leveraging its diverse libraries and frameworks. The Google Colab Pro environment, equipped with a substantial V100 gpu, 70GB drive or more drive, gpu ram at least 16GB and system ram greater than 40GB, provides a robust platform for the development and deployment of the AI chatbot system. The development environment for building the AI chatbot system platform is structured as follows:

- Operating System: Google Colab Pro
- Central Processing Unit (CPU): Not explicitly mentioned, but assumed to be part of the Google Colab Pro environment.
- GPUs (Graphics Processing Units): at least 16GB
- Random Access Memory (RAM): greater than 40GB
- Programming Language: Python

Install the necessary libraries for the project including the Ubuntu virtual machine, the Langchain framework will be the coordination framework for the project, Chroma DB will be the place to store the vector store and other necessary libraries.

```
!pip install -qqq torch==2.1.0
!pip install -qqq langchain==0.0.266
!pip install -qqq chromadb==0.4.5
!pip install -qqq xformers==0.0.20
!pip install -qqq sentence_transformers==2.2.2
!pip install -qqq InstructorEmbedding==1.0.1
!pip install -qqq websocket-client
!pip -qqq install gradio
!pip install -qqq python-engineio==3.14.2 python-socketio==4.6.0
!wget -q
https://github.com/PanQiWei/AutoGPTQ/releases/download/v0.4.1/auto_gptq-
0.4.1+cu118-cp310-cp310-linux_x86_64.whl
!pip install -qqq auto_gptq-0.4.1+cu118-cp310-cp310-linux_x86_64.whl
!sudo apt-get install poppler-utils
```

```
import torch from auto_gptq
import AutoGPTQForCausalLM from langchain
import HuggingFacePipeline, PromptTemplate
from langchain.chains import RetrievalQA
from langchain.embeddings import HuggingFaceInstructEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import DirectoryLoader, TextLoader
from langchain.vectorstores import Chroma
from transformers import AutoTokenizer, TextStreamer, pipeline
import locale
import re
import requests
import json
import time from datetime
import datetime DEVICE = "cuda:0" if torch.cuda.is_available() else "cpu"
```

First, we implement the library necessary for the project. Data collection is very important in building AI chatbots. Here we need to collect data and build a dataset according to the Us stock market sector from the website US stock. We use Websocket methods to collect the current price of the stock because the stock changes every second. In addition, we can use the API Restful to collect the data we need to collect in website US stock.

After collection, the data will be in json file format and this data will be assigned to the “Dataset.txt” file that we have created to create semantics from this data. For example, follow this format to get the current price of stock “Current price of {Symbol} is {current_price}.”. Following this, a dataset containing current stock prices will be gathered using statements like "Current price of AAPL is 177.97". Additionally, datasets pertaining to financial data and option data will be created in alignment with the system's requirements. After collecting data into txt files such as: Option.txt, Price.txt, Financials_data.txt, load the txt files into documents. 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()
```

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. The effectiveness of this process depends on the nature of the data and the specific requirements of the task at hand.

```
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)
```

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, offering utility in tasks such as retrieval, classification, clustering and semantic search. Additionally, these embeddings can be utilized in vector databases for Large Language Models and stored in a vector store. In this context, Chroma DB serves as the designated vector store for our purposes.

```
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)
```

After embedding the data, we store it in Chroma DB, which serves as a vector database to form the base knowledge for the chatbot system. Initially, we must persist this Chroma DB so that the chatbot can respond to questions based on this foundational knowledge. However, it is necessary to update the base knowledge because some data may change. For example, stock prices in the U.S fluctuate every second, forcing us to use websockets to collect real-time data. 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.

After retrieving data in real time, putting it into a txt file, we will read and process the data, divide it into appropriate parts and update. Updating based on the id and change data of the US stock price, we will determine the id of the vector to update in the database and change the document of that vector with the document of the US stock price in real time.

```
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])
```

As for the system part we can see in *Figure 3*, in the LLM part we use TheBlocke/Llama-2-13-B-chat-GPTQ to be able to generate answers word to users. This repo contains GPTQ model files for Meta's Llama 2 13B-chat.

```
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
```

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. Here we aim for an answer that is helpful, respectful and honest. In cases where the question is unreasonable or illogical, the model should explain why instead of answering incorrectly. When the model doesn't know the answer, it should admit this instead of trying to give wrong information.

```
DEFAULT_SYSTEM_PROMPT = """ You are a helpful and respectful
assistant committed to providing honest responses. Your answers should be
positive, safe, and free from harmful or inappropriate content.
If a question is unclear or factually incorrect, your approach is to explain the
issue rather than providing inaccurate information. If you don't know the
answer, you refrain from sharing false information.. """.strip()
def generate_prompt(prompt: str, system_prompt: str =
DEFAULT_SYSTEM_PROMPT) -> str:
   return f"""
  [INST] <<SYS>>
  {system_prompt}
  <</SYS>>
  {prompt} [/INST]
  """.strip()
```

```
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, )
```

Next we convert the text into numbers and configure text generation using the NLP model. Specific parameters include: "Text-generation": pipeline type, indicates you are using model to generate text, "model" NLP model used to generate text, "tokenizer" used to convert text to number for the model, "max_new_tokens" limits the number of new tokens created in a reply. If exceeded, the answer may be truncated. The "temperature" parameter defines the level of creativity of the answer. Low values (close to 0) produce more specific answers, while high values (e.g. 1) produce more creative answers. The "top_p" parameter governs the selection of tokens based on probability . This value limits token selection based on probability until the total probability exceeds this value, the "repetition_penalty" parameter governs the repetition of tokens in the answer. We create a configuration to use Hugging Face's NLP model to generate text with custom parameters such as creativity level, token count limit and other parameters to control the text generation process .

```
streamer = TextStreamer(tokenizer, skip_prompt=True,
skip_special_tokens=True)
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})
```

Next we load, split the text, process the data, represent the text data as arithmetic vectors and save them in vector storage and build a process for retrieving information and answering questions based on 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}, )
```

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)
```

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. The first way is "Embed Gradio UI with iframe", step 1 is to create an HTML bar in the website, step 2 in that HTML embed the <iframe> tag to embed Gradio UI into the website.

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

The second way to integrate Gradio into your website can be using Ajax technology. It is an abbreviation for the phrase "Asynchronous JavaScript and XML". This is a web programming technique that allows data to be transmitted and received from the server without reloading the entire web page. Ajax helps increase user experience by making websites faster and more flexible. Step 1 also creates an HTML bar in the website, step two uses Ajax to communicate with Gradio UI. Finish part 3 and continue to part 4 case study showing you how to apply the AI chatbot system to specific case studies.

```
var xhr = new XMLHttpRequest();
   xhr.open('GET', 'https://your-gradio-url', true);
   xhr.onload = function() {
      if (xhr.status === 200) {
            var gradioUI = xhr.responseText;
            document.getElementById('gradio-container').innerHTML =
gradioUI;
         }
      };
   xhr.send();
```

IV. CASE STUDY AND DISCUSSION

4.1 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, it includes many functions, all functions are built around stocks in the US. The company asked us to build a chatbot to answer questions about information related to US stocks on the website stockscan.io to help increase the customer experience so they can experience the service in an enjoyable way. The functions are listed below described in ***Table 5.***

Table 5. Component of the website stockscan.io US stock

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.

Stock Calculator	Calculate average stock price, calculate profit and loss of stock investment.
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)

After reviewing the functions of a US stock website, we need to learn about what customers are interested in the US stock market field when accessing the website, thereby providing the main answering functions of the chatbot for a website. According to the experts of the website, customers will be interested in the current price of stocks, the option function of the website, key stats of stocks and financial data (revenue, net income, cash flow, eps, debt to equity ratio) described in ***Table 6.***

Table 6. Data parameters needed by the stockscan.io

Data parameters	**Define**
Stocks	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.

Here we need to collect data and build a dataset according to the Us stock market sector from the Stockscan.io. 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). Then attach them to the available dataset and save it as a txt file. Example gathering data about financial data in Stockscan.io:

```
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"]
```

In addition to collecting data using RestfulAPI, 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'])
```

Finally, we apply section *3.2. Implementation* above to build an AI chatbot for the website stockscan.io described in **Fig .7.** There are a few questions users can ask the chatbot such as: "Revenue of AAPL in Q2 of 2023?" to ask about financial data of Apple company, "Current price of AAPL?" to ask about Apple's current stock price, etc. Answers will be given by the chatbot in about 3-5 seconds.

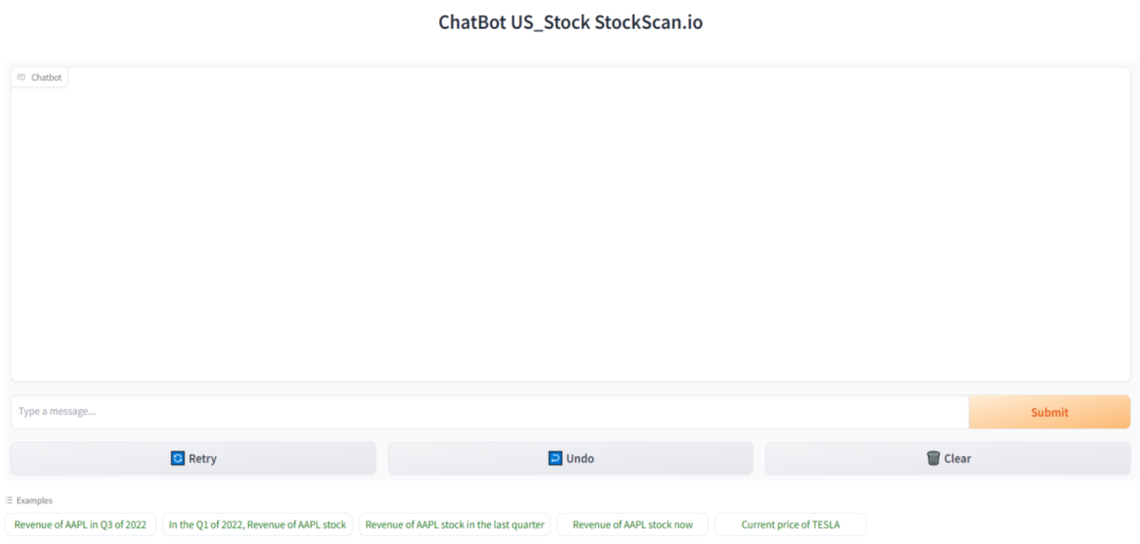

ChatBot US_Stock StockScan.io
Chatbot
Type a message...
Submit
Retry
Undo
Clear
Examples
Revenue of AAPL in Q3 of 2022
In the Q1 of 2022, Revenue of AAPL stock
Revenue of AAPL stock in the last quarter
Revenue of AAPL stock now
Current price of TESLA

Fig .7. Stockscan.io Chatbot

4.2 Discussion

In this part, after building an AI chatbot for stockscan.io, we tried many question sets to test this chatbot. The answers are highly accurate when the questions are related to the data in ***Table 6*** and the information of the functions that the chatbot can answer mentioned in *section 4.1*. 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. Other performances will be mentioned in ***Table 7.***

Table 7. Performance of chatbot AI stockscan.io replied 2000 US stock

	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

Nevertheless, there are instances where the chatbot's comprehension and accuracy are compromised, particularly when questions are not presented in the expected format. It is crucial to phrase questions in a specific manner for optimal results. For instance, the chatbot can successfully respond to inquiries like "Revenue of AAPL in Q2 of 2023", but may struggle with non-standard formats such as "Revenue of APPLE now" due to a lack of understanding. This limitation arises from the absence of question and answer formats like "Revenue of APPLE now" in our training dataset.

There are many ways to solve this problem, but we will give 2 ways to solve the problem above. The first way is that you can use more powerful embedding models, more powerful LLM models related to text generation to finetune your data set like GPT-4, LLAMA2-70b-chat-hf,..etc. You can improve your chatbot quickly, but the cost issue is something you need to consider because it costs a lot of money when used for small businesses. The second way is to create a larger answer data set with a more diverse format. This way you need to clearly understand your data and depend on the requirements of the business.

V. CONCLUSION AND PERSPECTIVES

In the endeavor to develop an AI chatbot tailored for a US stock website, we laid the groundwork with the RAG method and the Langchain framework. This foundational chatbot exhibits proficiency in addressing inquiries related to US stocks, encompassing functionalities such as retrieving current prices, providing financial data. Specifically designed for small-sized businesses, this platform aims to elevate the overall customer experience.

Throughout the project implementation, my team gained valuable insights into well-established chatbot creation technologies like Rasa and explored tools facilitating code-free chatbot development, utilizing prominent large language models such as Botpress and Stack AI. Additionally, delving into the creation of a chatbot for the US stock industry enriched our understanding of various concepts within this domain.

However, it's crucial to recognize that this marks only the foundational structure of the chatbot system. There exists ample room for refinement and enhancement, allowing the chatbot to handle a broader array of questions and deliver more robust responses. Furthermore, there is potential to introduce novel features, such as investment advisory support. By leveraging data and statistics, the chatbot can provide valuable investment advice, assisting customers in gaining a deeper understanding of their investment options.

VI. REFERENCES

1. Cheonsu Jeong. A Study on the Implementation of Generative AI Services Using an Enterprise Data-Based LLM Application Architecture. 2309.01105.pdf. 2023.
2. Myeong-Ha Hwang, Jikang Shin, Hojin Seo, Jeong-Seon Im, Hee Cho. ChatRPA: Open Source-Based Conversational Chatbot System for Robotic Process Automation. 2022.
3. Yeon Seonwoo, Juhee Son, Jiho Jin, Sang-Woo Lee, Ji-Hoon Kim , Jung-Woo Ha, Alice Oh. Two-Step Question Retrieval for Open-Domain QA. 2205.09393v1.pdf. 2022.
4. Jeppiaar Nagar, Rajiv Gandhi Salai, Chenna. Herbivicus: A full stack website with chatbot and google API. 2021.
5. Isabelle Augenstein, Timothy Baldwin, Meeyoung Cha, Tanmoy Chakraborty. Factuality Challenges in the Era of Large Language Models. 2310.05189v2.pdf. 2023.
6. Deussom Djomadji Eric Michel College of Technology University of Buea Department of Electrical and Electronic Engineering. Design and Implementation of a Chatbot for the Supervision of Security Events (SIEM).2023.
7. Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem. Emad Shihab Challenges in Chatbot Development: A Study of Stack Overflow Posts. October 5–6, 2020.
8. Adith Sreeram A S, Pappuri Jithendra Sai. An Effective Query System Using LLMs and LangChain. June 2023.
9. Hana Demma Wube, Sintayehu Zekarias Esubalew, Firesew Fayiso Weldesellasie and Taye Girma Debelee. Text-Based Chatbot in Financial Sector: A Systematic Literature Review. 2022.
10. Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa , Matt Beane. FINQA: A Dataset of Numerical Reasoning over Financial Data. 2109.00122v3.pdf. 2021.
11. Qianqian Xie, Weiguang Han, Xiao Zhang, Yanzhao Lai, Min Peng, Alejandro Lopez-Lira, Jimin Huang. PIXIU: A Large Language Model,

Instruction Data and Evaluation Benchmark for Finance. 2306.05443v1.pdf. 2023.

12. Rudi Setiawan, Rossi Iskandar, Nadilla Madjid, Ridwan Kusumawardan. Artificial Intelligence-Based Chatbot to Support Public Health Services in Indonesia. 2022.
13. James C. L. Chow , Valerie Wong , Leslie Sanders, Kay Li. Developing an AI-Assisted Educational Chatbot forRadiotherapy Using the IBM Watson Assistant Platform.IBM Watson Assistant. 2023.
14. Shamane Siriwardhana, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, Suranga Nanayakkara. Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering. 2210.02627v1.pdf. 2023.
15. Shuo Li, Sangdon Park, Insup Lee, Obsert Bastani. TRAC: Trustworthy Retrieval Augmented Chatbot. 2307.04642v1.pdf.2023.
16. Nura Esfandiari , Kourosh Kiani , Razieh Rastgoo. A Conditional Generative Chatbot using Transformer Model. 2306.02074.pdf.2023.
17. Jiangtong Li, Yuxuan Bian, Guoxuan Wang, Yang Lei, Dawei Cheng, Zhijun Ding, Changjun Jiang. CFGPT: Chinese Financial Assistant with Large Language Model. 2309.10654v2.pdf. 2023.
18. Hugo Touvron, Louis Martin, Kevin Stone. Llama 2: Open Foundation and Fine-Tuned Chat Models. Llama 2. 2023.
19. Kim Martineau: What is retrieval-augmented generation?.RAG. 2023.
20. Jiawei Chen, Hongyu Lin, Xianpei Han, Le Sun. Benchmarking Large Language Models in Retrieval-Augmented Generation. 2309.01431v1.pdf. 2023.