Leveraging LLM for News Retrieval of the Thailand Stock Market

Serena Banghar, Qin Dai, Alina Hasan, Pinzhe Liu

28 April 2025

Contents

1 Introduction

Thailand's stock market, led by the Stock Exchange of Thailand (SET), plays a key role in the country's economy and offers investors a diverse range of investment opportunities across several industries. This project presents the development of a specialized chatbot designed to help users gain a better understanding of Thailand's stock market by providing access to real-time data and market insights. By leveraging Google's Gemini language model, the chatbot can understand and respond to user queries with relevant and informative answers. To keep financial insights accurate and current, the chatbot is integrated with Yahoo Finance, ensuring users receive up-to-date stock data specific to the Thai market.

The backend of the chatbot is built using FastAPI, enabling efficient handling of API requests, management of conversation history and retrieval of relevant data. A user-friendly frontend, constructed with HTML and JavaScript, allows users to easily submit questions and view responses on the web interface. By combining generative AI with real-time market data, the chatbot delivers quick, informed guidance for anyone looking to better understand the Thai stock market.

2 Data Sources

2.1 LLM

Large Language Models (LLMs) are artificial intelligence models trained on large amounts of text data to comprehend human languages. These models rely on neural networks, such as transformers, to interpret the meaning and context between words and phrases[1]. By learning grammar and language semantics, LLMs can predict the next word in a sentence, fill in missing words or check if one sentence logically follows another. Common examples of LLMs include Open AI's GPT and Google's BERT. LLMs are utilized across various industries, including finance, where they help assess news articles to predict stock market trends[2].

2.2 Google Gemini

Gemini is built on a large language model, allowing it to effectively understand the nuances in user queries. Gemini combines its language capabilities with reasoning and multi-modal processing to understand complex queries and provide insightful responses, enhancing user interactions[3].

2.3 Yahoo Query

The yahooquery library delivers near-real-time and historical stock-market data, including price movements, trading volumes, financial ratios, and other key metrics that investors rely on for in-depth company analysis. Its API and Python integration allow users to build custom financial models and automated trading strategies, supporting informed decision-making.

2.4 News Source

The chatbot utilizes the Bangkok Post, Thailand's leading English-language newspaper, as a reputable news and financial source to deliver accurate insights into developments within the Thailand stock market. Reports from the Bangkok Post's business desk covers several key insights including daily market movements, sector changes, regulatory updates and corporate earnings. The Bangkok post also consults published materials from exteneral news organizations to provide broader context for local movements. By

integrating Bangkok Post commentary, the chatbot delivers well-rounded answers that reflect the current conditions in Thailand's stock markets.

3 Thailand Stock Market

The Thailand stock market primarily trades on the Stock Exchange of Thailand (SET), which covers a wide range of sectors including resources, financials, consumer products and services [4]. Over recent years, Thailand stocks have become increasingly attractive to foreign investors, due to its strategic location and adoption of government initiatives aimed at economic development. The market is regulated by the Securities and Exchange Commission (SEC) of Thailand, ensuring transparency and protection for investors. Major indices include the SET50 and SET100, which are composed of the largest 50 and 100 companies by market capitalization, respectively, and act as benchmarks for market performance [4].

In 2024, Thailand's stock market experienced significant downturns. However, the market began to recover in the latter half of the year, indicating resilience and potential for future growth [5]. The beginning of 2025 appeared promising, as the market was boosted by government support and the resurgence of tourism. Government initiatives, such as providing tax benefits for foreign investors and enforcing stricter regulatory procedures, are expected to encourage future investment [5,6,7].

The Thailand equity market is dominated by several large-capitalization companies representing key sectors of the economy, whose performance disproportionately impacts index trends. PTT Public Company Limited (PTT) is a major part of the energy sector, with involvement in all stages of the oil and gas process from sourcing to production, and active expansion into renewable energy. Delta Electronics (Thailand) Public Company Limited (DELTA) is a global provider of power management for data centers and electric vehicle infrastructure. Advanced Info Service Public Company Limited (AIS, ticker ADVANC) provides the nation's largest telecommunications network and is utilizing its 5G technology to expand into cloud gaming, streaming and financial technology platforms. Airports of Thailand Public Company Limited (AOT) manages six international airports, including Suvarnabhumi in Bangkok, and serves as an indicator for tourism performance. CP All Public Company Limited (CPALL) maintains a dominant market position in convenience retail through its 7-Eleven franchise network.

Gulf Energy Development Public Company Limited (GULF), is Thailand's fastest-growing independent power producer, with investments in liquified natural gas (LNG) infrastructure and offshore wind energy. Systemically important banks in the financial sector include SCB X, Bangkok Bank (BBL), and Kasikornbank (KBANK). They provide corporate financing and digital banking services across the southeast asian region.

In addition to equity trading, the Thailand Futures Exchange (TFEX), a SET subsidiary, offers futures and options based on the SET50 index. These contracts are settled in cash with a multiplier of THB 200 and monthly and quarterly maturities. TFEX also offers single-stock futures for more than 140 stocks, including all companies in the SET50. Additionally, there are sector-specific index futures for industries such as banking, energy and commerce, as well as contracts for commodities and currencies, including gold, silver, rubber, and foreign exchange pairs like USD/THB, EUR/THB and JPY/THB. All instruments are cleared through the Thailand Clearing House with daily margining. Both domestic and foreign investors can access these instruments through licensed Thai brokers.

Together, these markets allow investors to hedge their investments with SET50 futures, take advantage of temporary price differences through trades between single-stock futures and their underlying stocks, and express views on trends through investing in specific stocks or sector-index futures.

Additionally, global factors such as fluctuating U.S. interest rates, ongoing trade tensions, and geopolitical conflicts could impact market stability. However, Thailand's strong economic environment, supported by tourism, exports, and manufacturing, make it a promising market for long-term investors who remain well-informed about the evolving investment landscape [5,6,7].

4 Tools

4.1 FastAPI

FastAPI is a modern framework for building application programming interfaces (APIs) in Python. By making Python's built-in type hints a core feature, the system uses Pydantic to automatically check and verify data being sent and received, and it creates interactive documentation for the API (Swagger UI and ReDoc) without needing additional coding. FastAPI

is built using Starlette, a toolkit designed to handle many tasks at once, and is powered by Uvicorn, an engine that supports handling multiple connections simultaneously. This allows FastAPI to manage several requests concurrently, making it ideal for processing real-time data.

FastAPI has a system that helps organize and manage different parts of the application in a way that makes it easy to test and update. It also allows certain tasks, such as logging and analytics, to run in the background without interfering with the app's main functions. Security features are embedded, such as authentication protocols and limiting request frequency, to help developers comply with industry standards.

In this project, FastAPI serves as the backbone of the backend system, managing user requests and facilitating the integration of real-time data. Its performance ensures efficient processing of user queries and integration of live market data. Further, FastAPI's automatically generated interactive documentation aids in developer productivity by simplifying testing and improving API understanding, leading to decreased time spent on debugging and repetitive coding. Overall, FastAPI's simplicity, speed, and reliability makes it an ideal choice for our Thailand Stock Market Chatbot [8].

4.2 RAG

Retrieval-Augmented Generation (RAG) enhances the output of LLMs through combining real-time information retrieval with natural language processing (NLP) techniques to deliver answers based on sourced information. In the first stage, a retrieval module searches external sources, such as websites, scientific literature or databases, to find content that is relevant to the user's query. In the second stage, the LLM utilizes both the retrieved materials that displayed the most similarity to the user's query and the original question to generate an informational response that cites the retrieved evidence. RAG identifies relevent information through key word identification. This two-step process mitigates the risk of an LLM providing inaccurate information by verifying the model's generated response with supporting documentation. As a result, RAG is valuable in building applications that answer questions, perform searches or summarize information. The RAG system is continuously improving through better result ranking and more efficient shortening of the retrieved text, enhancing overall performance [9].

5 System Overview

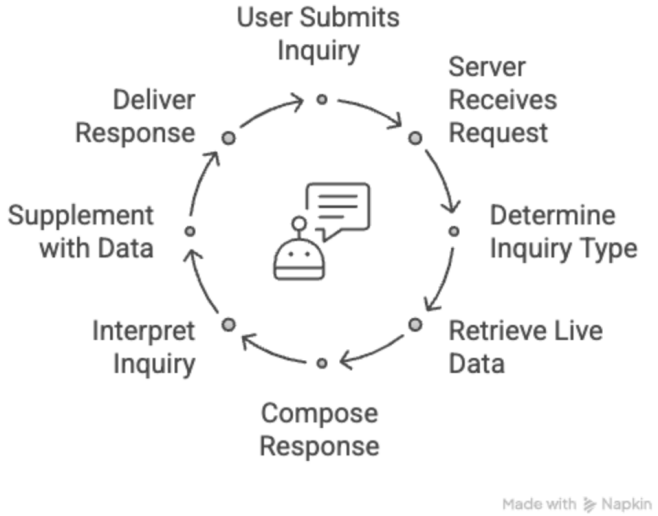

Figure 1: Chatbot inquiry process from user input to response delivery.

The system is a microservice built with FastAPI that combines real-time Thai stock prices with large language model reasoning. When a user sends a request, a rule system is used to determine whether a numerical quotation or explanation is needed. If the request is numerical in nature, the system uses the yahooquery library to fetch the latest data. For more complex inquiries, the system constructs a structured prompt and forwards it to Google Gemini for response generation. If the user needs both numerical data and an explanation, the system combines real-time data with generated text to create a complete response.

The response is returned in a single JSON file, making it easy for the chatbot to use. In order to maintain continuity throughout the conversation, the system temporarily saves the session's transcript, allowing follow-up questions to be answered with full context. Error handling is implemented throughout and all external API calls are logged for monitoring and troubleshooting. This makes the chatbot reliable and easy to maintain.

Figure 1 presents the closed-loop sequence that transforms a user's question into a response. Each stage has a distinct purpose in providing an ac-

curate, detailed answer.

1. User Submits Inquiry → Server Receives Request

When a user submits an inquiry through the chatbot's web interface, the request is sent to the backend API. The server then assigns the message a unique ID, adds important data such as a timestamp and session information, and appends the information to a conversation log. The full dialogue is included with every new query to ensure the chatbot interprets each message within the correct context, allowing for more accurate responses.

2. Determine Inquiry Type

A simple rule-based system is used to extract key information from the inquiry such as ticker symbols, phrases like "price of" or "latest quote", or time words like "now" or "today." Based on the nature of the request, the server chooses an execution path: real-time data retrieval or an AI generated response.

3. Retrieve Live Data

If the user's inquiry requires numerical data, such as "What is the price of PTT?", the system queries a reliable live market data source to retrieve the latest metrics. The server then creates a short, fact-based reply and returns it back to the user without using the language model for processing. This decreases the time necessary to generate an appropriate response and the risk of providing incorrect numbers.

4. Generated Response

For inquiries that require interpretation, such as "What factors are driving market movements?", the server combines three components into a single prompt: a standing instruction directing the system to respond as a Thai market adviser, the saved dialogue history, and the new question. This composite prompt is then sent to Google Gemini which generates a coherent explanation.

5. Interpret Inquiry and Supplement with Data

When necessary, the server supplements Gemini's output with relevant numerical data before delivering the final response to the user. The addition of factual information supports the commentary to increase reliability of the response.

6. Deliver Response

The final response is formatted and sent back to the frontend through the HTTP connection. This entire process, from query receival to response delivery, typically occurs within a few seconds, providing users with a timely, response that effectively integrates live market data with commentary.

6 Comprehensive Description

The `chatbot_gemini.py` module forms the backend logic for the web-based Thai stock market chatbot. Implemented using FastAPI, the system provides three endpoints `/chat GET, /chat POST, /realtime_index` and one informational landing page. The code integrates three external resources, Google Gemini, Yahooquery, and the Stock Exchange of Thailand (SET) Marketplace API, while maintaining conversational context in memory. Its structure and operational flow are detailed below.

6.1 Imports and Initialization

FastAPI primitives (`FastAPI, Query, Body, HTTPException`) help route requests and ensure proper data validation.
CORSMiddleware ensures the application can accept requests from various sources during development.
Requests is used to handle synchronous calls to Gemini and the SET Marketplace.
yahooquery supplies live closing prices for Thailand stocks.
uvicorn handles incoming HTTP requests and manages application execution.
logging module-level logger is set up with the `INFO` logging level, ensuring information about external calls is recorded and any exceptions are caught, along with time stamps.

6.2 Global Constants and Session Store

`GEMINI_API_KEY, SET_API_KEY` are hard-coded credentials that enable outbound requests.
`SYSTEM_INSTRUCTION` is a standing prompt that frames every Gemini interaction: "You are a financial assistant specializing in the Thai stock market."
`conversation_history` is a dictionary that maps `session_id` strings to concatenated question-answer transcripts. It enables continuity throughout the dialogue.

6.3 Outbound Service Wrappers

`call_gemini_api(prompt:str)` constructs a JSON file consistent with the format that Gemini's *generateContent* function requires, posts to the `gemini-2.0-flash` endpoint and parses the response, enforcing four checks (valid JSON, non-empty candidates, non-empty content, non-empty text part). Finally, it either returns the first candidate's raw text or raises an `HTTPException(502)` if there is a structural error.
`get_stock_price_yq(symbol:str)` uses *yahooquery.Ticker* to pull one-day price history. If there is no data, it returns an apology phrase. On success, it extracts the `regularMarketPrice`, formats a sentence, and logs any errors.
`extract_symbol_after_trigger(query:str)` converts the query to lowercase for case-insensitive matching and scans for trigger phrases such as "price of" or "quote for". If these phrases are found, it isolates the first subsequent token, converts it to uppercase, and appends the Bangkok suffix ".BK". It returns `None` if no trigger is present.

6.4 Route Definitions and Control Flow

`GET/` returns a welcome message.
`GET /chat`is a stateless query path. It concatenates `SYSTEM_INSTRUCTION` with the user's query, calls Gemini, and returns the model's reply without storing the conversation state.
`POST /chat` is a stateful conversational path with three stages. Price-trigger shortcut executes `extract_symbol_after_trigger`. If a ticker is detected, it calls textttget_stock_price_yf and responds immediately. Contextual prompt assembly occurs when no trigger is found. It retrieves the existing transcript (or the default system instruction), appends the latest question under a `User:` label, and cues `Assistant:` for Gemini. Transcript update stores the prompt plus Gemini's answer back into `conversation_history` to preserve continuity. Each branch returns a JSON object containing both `session_id` and the finished `response`.
`GET /realtime_index` is a parameterised proxy to the SET Marketplace "realtime-data/stock" endpoint. The query returns index snapshots for the SET, sector aggregates (e.g., SET50, BANK), and three well-known symbols (PTT, AOT, EGCO). Raw JSON data from the exchange is forwarded to the caller, with the HTTP status and body provided in the event of an error.

6.5 Error Handling and Observability

All external requests are wrapped in `try/except blocks`. Network or parsing failures generate descriptive `HTTPException` responses and concurrent log entries, ensuring that client applications receive status codes while developers can trace issues.

6.6 Run-time Environment

When the module is invoked as a script, `uvicorn.run()` launches an ASGI instance on `127.0.0.1:8000`. FastAPI requests are asynchronous, allowing a single worker to handle multiple requests at the same time.

6.7 Security and Scalability Considerations

API keys are currently hardcoded, but in production, they should be sourced from environment variables for effective credential management.
Using an in-memory dictionary is sufficient if the chatbot runs on a single server. If the chatbot is run on multiple servers, it requires an external storage system (like Redis) to ensure data consistency.
Setting `allow_origins=["*"]` in CORS policy simplifies testing, but is not secure. Before public release, API access should be restricted to trusted domains.

6.8 Functional Summary

The module delivers two complementary behaviours. If the user writes "price of PTT", the assistant extracts "PTT.BK", fetches `regularMarketPrice`, and replies instantly. All other queries go through Gemini with the running transcript, enabling conversational explanations.

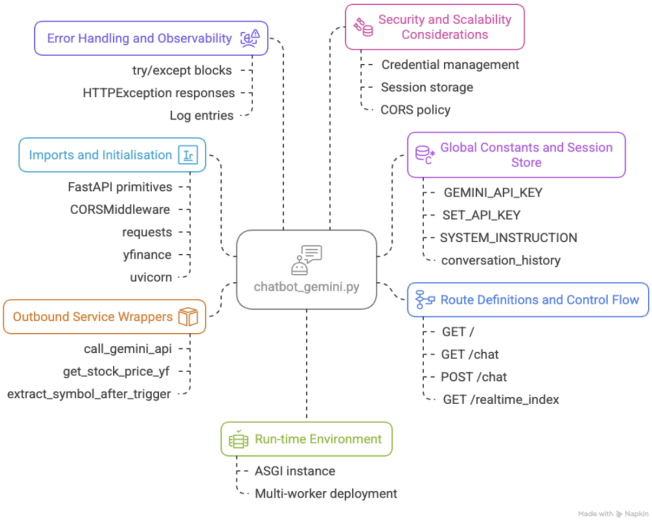

Figure 2: Architecture of `chatbot_gemini.py`

7 Results

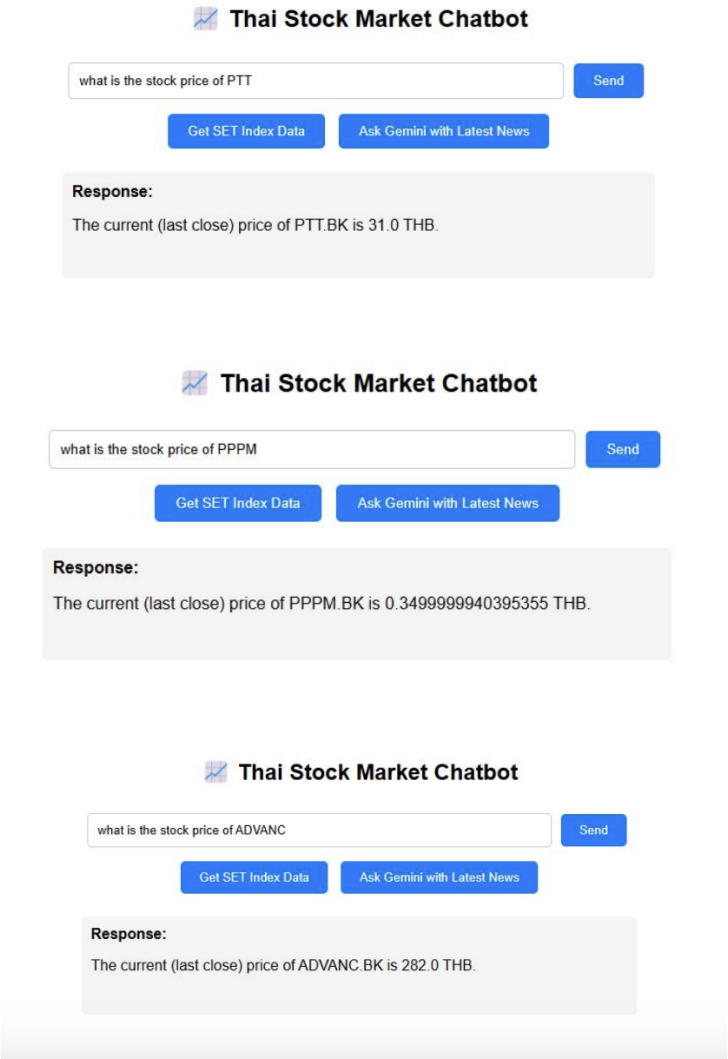

Figure 3: Sample queries of chatbot externally sourcing stock market data.

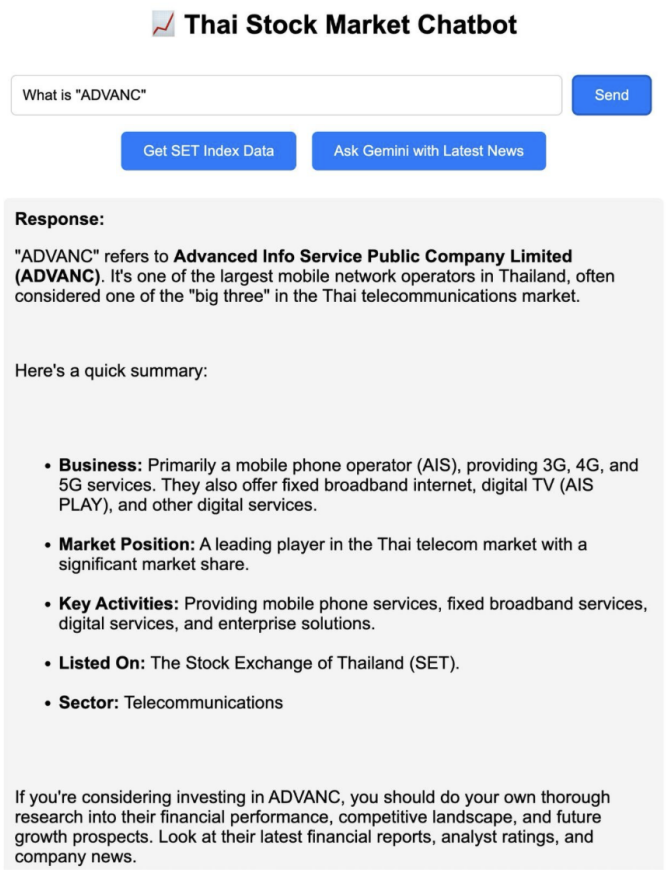

Figure 4: Sample query of chatbot answering explanatory question through sweep of online news services. The chatbot provides key features including business information, market position, key activities and sector.

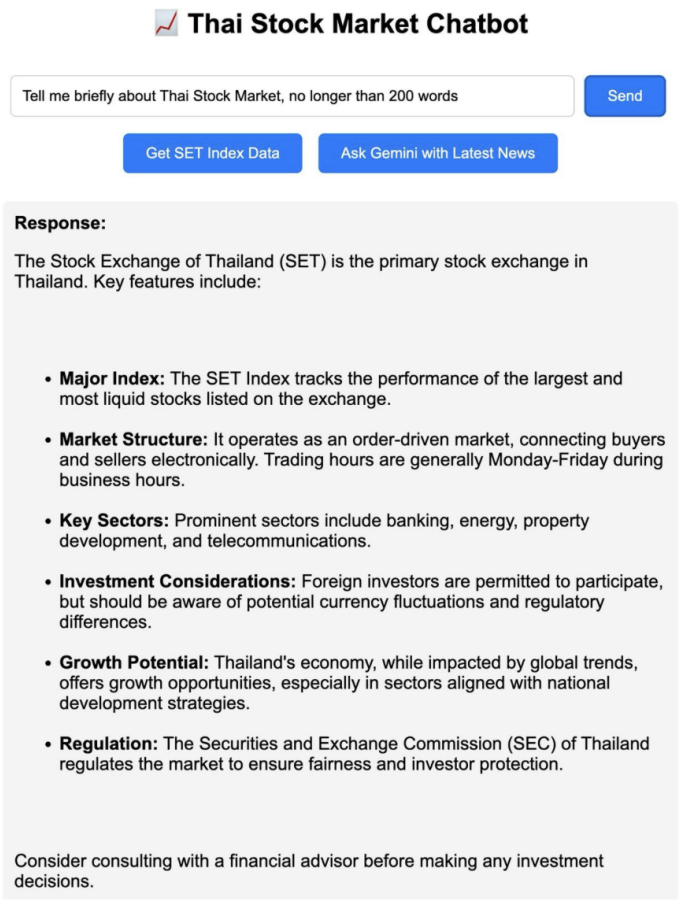

Figure 5: Sample query of chatbot providing nuanced response with set parameters. Key features are mentioned, including description of the major index, market structure, sectors and growth potential, within a 200 word set limit.

Figure 3 demonstrates the output of the chatbot when a numerical question is asked. A response is generated without consulting Gemini, resulting in a short, formatted sentence. **Figure 4** displays the chatbot output when the user inputs an explanatory question. A response is generated based on key word identification from articles previously collected by the scraper, and formated in Gemini to create a cohesive response. **Figure 5** demonstrates the chatbot's ability to create responses with additional rules such as a specified word count.

8 Further Work

The chatbot prototype shows that the large language model combined with direct price feeds can effectively answer many routine Thai Stock Market Chatbot queries. However, the system fails to answer questions that require additional background information that is not available through the Bangkok Post or a Yahoo Finance stock quote. Addressing these concerns requires two additions to the chatbot.

First, we will build a scraper that runs on a rolling schedule across several additional news outlets, including Thailand specific news sources such as Thailand Business News or the official website of the Stock Exchange of Thailand (SET), and global news outlets such as Reuters and Bloomberg. Each article will be tagged with source information and stored in a database, making new facts searchable shortly after they are published. By automating data collection rather than relying on manual uploads, we create a constantly updated system that keeps up with Thailand's rapidly changing economic landscape.

Initial experiments with RAG revealed its limited accuracy when the article database is small or the system cannot find a match for keywords from the user's queries. This results in Gemini generating answers that are incorrect to fill in the missing information. To improve the accuracy of the system we will introduce confidence-gated routing, which introduces a confidence threshold to decide next steps during the retrieval process. If the retrieved data meets a similarity threshold, the passage will be forwarded to Gemini and the answers will be cited. Otherwise, the Agent-only pathway will be used to avoid making up false references. Initial tests show that combining several relevant passages with the Agent's reasoning helps reduce fabricated information without largely increasing response time.

Together, the automated content collection and confidence-based hybrid RAG system will transform the chatbot from its current prototype to a fully functional assistant that can deliver real-time and source-backed insights across Thailand's markets. Finally, we will track detailed metrics like retrieval success rate and response time to ensure the improvements are based on objective metrics as opposed to informal testing.

9 Works Cited

1. *What is LLM (Large Language Model)?*.AWS.(n.d.). https://aws.amazon.com/what-is/large-language-model/
2. Miller, J. (2025).*NLP Part II- LLMs and Financial Applications* [Lecture notes].Canvas. https://canvas.jhu.edu/
3. Pichai, S. (2023, December 6).*Introducing Gemini: Our largest and most capable AI model*. Google. https://blog.google/technology/ai/google-gemini-ai/sundar-note
4. *SET index overview - the Stock Exchange of Thailand*. Index Overview - The Stock Exchange of Thailand. (n.d.). https://www.set.or.th/en/market/index/set/overview
5. Sullivan, B. (2025, January 12).*Key statistics of the Thai Stock Exchange: December 2024 and Outlook for 2025*. Thailand Business News. https://www.thailand-business-news.com/set/188013-summary-of-important-statistics-of-the-thai-stock-exchange-in-december-and-2024
6. Prakash, A. (2024, December 4). *8 stocks in Thai Market to Benefit from Headwinds in 2025*. Thailand Business News. https://www.thailand-business-news.com/markets/178423-8-stocks-in-thai-market-to-benefit-from-headwinds-in-2025
7. *Investing in... 2025*. Chambers and Partners.(n.d.). https://practiceguides.chambers.com/practice-guides/investing-in-2025/thailand
8. Yadav, N. (2024, January 8). *Introduction to FastAPI*. Medium. https://medium.com/@yaduvanshineelam09/introduction-to-fastapi-123c0b2778a5
9. Wikimedia Foundation. (2025, April 22). *Retrieval-augmented generation*. Wikipedia. https://en.wikipedia.org/wiki/Retrieval-augmented_generation

10 AI Disclosure

AI was utilized to help write and develop the following sections in addition to using online articles: Data Sources, Thailand Stock Market, Tools, System Overview and Comprehensive Description. It was further used to create Figure 1 and Figure 2.

11 Appendix

11.1 Github Link

https://github.com/Allan-dq/News-Retrieval-of-Thailand-Stock-Market/tree/main

11.2 Chatbot_gemini.py

```
from fastapi import FastAPI, Query, Body, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import requests
import uvicorn
import json
import logging
from typing import Optional
from yahooquery import Ticker

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("ThaiStockChatbot")

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

GEMINI_API_KEY = "AIzaSyAgol_MKg1rq8mJbJx3uMTYAlC1oYah8ZA"
SYSTEM_INSTRUCTION = "You are a financial assistant
↪  specializing in the Thai stock market."

SET_API_KEY = "0ab00c29-3df9-42b1-8325-56c15122f5e6"

conversation_history = {}
```

```python
def call_gemini_api(prompt: str) -> str:
    url =
    ↪ f"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash
    payload = {
        "contents": [{
            "parts": [{"text": prompt}]
        }]
    }
    headers = {"Content-Type": "application/json"}
    response = requests.post(url, headers=headers,
    ↪ json=payload)
    try:
        data = response.json()
    except json.JSONDecodeError:
        logger.error("Invalid JSON response from Gemini API.")
        raise HTTPException(status_code=502, detail="Invalid
        ↪ response from Gemini API.")

    candidates = data.get("candidates", [])
    if not candidates:
        logger.error("No candidates returned from Gemini API.")
        raise HTTPException(status_code=502, detail="No
        ↪ response candidates from Gemini API.")

    candidate = candidates[0]
    content = candidate.get("content", {})
    parts = content.get("parts", [])
    if not parts:
        logger.error("No text parts found in the candidate
        ↪ response.")
        raise HTTPException(status_code=502, detail="Incomplete
        ↪ response from Gemini API.")
    response_text = parts[0].get("text", "No text in the first
    ↪ part.")
    return response_text

def get_stock_price_yq(symbol: str) -> str:
```

```
    try:
        ticker = Ticker(symbol)
        # Get 1 day of historical data
        hist = ticker.history(period="1d")
        if hist.empty:
            return f"Sorry, I couldn't fetch the price for
            ↪ {symbol} right now."

        # 'Close' column will have the last available trading
        ↪ day's close price
        current_price = hist["Close"].iloc[-1]

        # Return a response string
        return f"The current (last close) price of {symbol} is
        ↪ {current_price} THB."
    except Exception as e:
        logger.error(f"Error fetching stock price for {symbol}:
        ↪ {str(e)}")
        return f"An error occurred while fetching data for
        ↪ {symbol}."



def extract_symbol_after_trigger(query: str) -> Optional[str]:
    triggers = [
        "price of",
        "current price of",
        "stock price of",
        "quote for",
        "value of"
    ]
    text_lower = query.lower()
    for trigger in triggers:
        if trigger in text_lower:
            # everything after the trigger
            after_trigger = text_lower.split(trigger,
            ↪ 1)[1].strip()
            if not after_trigger:
```

```
                return None
            # first word from that remainder
            first_word = after_trigger.split()[0]
            # assume it's a Thai ticker, append .BK
            return first_word.upper() + ".BK"
    return None

@app.get("/")
def welcome():
    return {"message": "Welcome to the Thai Stock Market
    ↪ Chatbot (Gemini)!"}

@app.get("/chat")
def chat_with_gemini(query: str = Query(...,
↪ description="User's stock market question")):
    try:
        combined_text = f"{SYSTEM_INSTRUCTION}\nUser:
        ↪ {query}\nAssistant:"
        response_text = call_gemini_api(combined_text)
        return {"response": response_text}
    except Exception as e:
        logger.error(f"Error in GET /chat: {str(e)}")
        return {"error": str(e)}

@app.post("/chat")
def chat_with_gemini_post(
    session_id: str = Query(..., description="Unique session
    ↪ identifier"),
    query: str = Body(..., embed=True, description="User's
    ↪ stock market question")
):
    try:
        # 1. Check if query includes a trigger phrase
        symbol = extract_symbol_after_trigger(query)
        if symbol:
            # 2. We found a symbol after the trigger => get
            ↪ stock price
            response_text = get_stock_price_yf(symbol)
```

```
            return {"session_id": session_id, "response":
            ↪  response_text}

        # 3. Otherwise fallback to Gemini
        history = conversation_history.get(session_id,
        ↪  SYSTEM_INSTRUCTION)
        updated_prompt = f"{history}\nUser:
        ↪  {query}\nAssistant:"
        response_text = call_gemini_api(updated_prompt)
        conversation_history[session_id] = f"{updated_prompt}
        ↪  {response_text}"
        return {"session_id": session_id, "response":
        ↪  response_text}
    except Exception as e:
        logger.error(f"Error in POST /chat for session
        ↪  {session_id}: {str(e)}")
        return {"error": str(e)}

@app.get("/realtime_index")
def get_set_realtime_data():
    url =
    ↪  "https://marketplace.set.or.th/api/public/realtime-data/stock"
    headers = {
        "api-key": SET_API_KEY
    }
    params = {
        "market": "SET,mai",
        "indexSector": "SET50,FINCIAL,BANK,INDUS-M",
        "securityType": "CS,DWC,DWP",
        "stockSymbol": "PTT,AOT,EGCO",
        "oddLotFlag": "false"
    }

    response = requests.get(url, headers=headers,
    ↪  params=params)
    if response.status_code == 200:
        return response.json()
    else:
```

```
        return {
            "error": response.status_code,
            "message": response.text
        }

if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

11.3 Chatbot_ui.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Thai Stock Market Chatbot</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 600px;
      margin: auto;
      padding: 20px;
      text-align: center;
    }
    input {
      width: 80%;
      padding: 10px;
      margin: 10px 0;
      border: 1px solid #ccc;
      border-radius: 5px;
    }
    button {
      padding: 10px 20px;
      background-color: #007bff;
      color: white;
      border: none;
      cursor: pointer;
      border-radius: 5px;
      margin: 5px; /* Give the button some spacing */
```

```
    }
    button:hover {
      background-color: #0056b3;
    }
    #response {
      margin-top: 20px;
      padding: 10px;
      background-color: #f4f4f4;
      border-radius: 5px;
      text-align: left;
      white-space: pre-line;
    }
  </style>
</head>
<body>
  <h2> Thai Stock Market Chatbot</h2>

  <input type="text" id="query" placeholder="Ask me about the
  ↪  Thai stock market...">
  <button onclick="sendMessage()">Send</button>
  <button onclick="fetchSetIndexData()">Get SET Index
  ↪  Data</button>
  <!-- New button -->
  <button onclick="sendNewsQuery()">Ask Gemini with Latest
  ↪  News</button>

  <div id="response"></div>

<!-- Import the marked library for Markdown to HTML -->
  <script
  ↪  src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>

  <script>
    let sessionId = localStorage.getItem("chatSessionId");
    if (!sessionId) {
      sessionId = "testsession-" + Date.now();
      localStorage.setItem("chatSessionId", sessionId);
    }
```

```
function sendMessage() {
  const query = document.getElementById("query").value;
  if (!query) {
    alert("Please enter a question.");
    return;
  }


  ↪  fetch(`http://127.0.0.1:8000/chat?session_id=${sessionId}`,
  ↪  {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({ query })
  })
  .then(response => response.json())
  .then(data => {
    console.log("API Response:", data);
    const responseElement =
    ↪  document.getElementById("response");
    if (data.error) {
      responseElement.innerHTML = `<strong>Error:</strong>
      ↪  ${data.error}`;
    } else {
      const rawMarkdown = data.response || "No response
      ↪  received.";
      const htmlContent = marked.parse(rawMarkdown);
      responseElement.innerHTML =
      ↪  `<strong>Response:</strong><br>${htmlContent}`;
    }
  })
  .catch(error => {
    console.error("Error:", error);
    document.getElementById("response").innerHTML =
      `<strong>Error:</strong> Unable to connect to the
      ↪  chatbot.`;
```

```
  });
}

function fetchSetIndexData() {
  fetch("http://127.0.0.1:8000/realtime_index")
    .then(response => response.json())
    .then(data => {
      console.log("SET Index Data:", data);
      const responseElement =
      ↪  document.getElementById("response");
      if (data.error) {
        responseElement.innerHTML = `<strong>API
        ↪  Error:</strong> ${data.error},
        ↪  ${data.message}`;
      } else {
        const jsonString = JSON.stringify(data, null, 2);
        responseElement.innerHTML = `<strong>SET Index
        ↪  Data:</strong><br><pre>${jsonString}</pre>`;
      }
    })
    .catch(error => {
      console.error("Error:", error);
      document.getElementById("response").innerHTML =
        `<strong>Error:</strong> Unable to fetch SET index
        ↪  data.`;
    });
}

// Added feature: first fetch the latest news data, then
↪  combine it with the user's question to call the Gemini
↪  API
function sendNewsQuery() {
  const query = document.getElementById("query").value;
  if (!query) {
    alert("Please enter a question.");
    return;
  }
```

```
// Call the SET news API, assuming the endpoint is
↪  /realtime_news
fetch("http://127.0.0.1:8000/realtime_news")
  .then(response => response.json())
  .then(newsData => {
    console.log("News Data:", newsData);
    let newsContent = "";
    if (newsData.error) {
      newsContent = "No news available.";
    } else {
    // Assuming the returned data contains a 'news'
    ↪  property, if not, convert the entire JSON
      newsContent = newsData.news ||
      ↪  JSON.stringify(newsData);
    }

    // Construct a combined query by integrating the
    ↪  latest news information with the user's question
    const combinedQuery = `${newsContent}\n\n${query}`;

    // Call the Gemini API to get the answer

    ↪  fetch(`http://127.0.0.1:8000/chat?session_id=${sessionId}`,
    ↪  {
      method: "POST",
      headers: {
        "Content-Type": "application/json"
      },
      body: JSON.stringify({ query: combinedQuery })
    })
    .then(response => response.json())
    .then(data => {
      console.log("Gemini Response with News:", data);
      const responseElement =
      ↪  document.getElementById("response");
      if (data.error) {
        responseElement.innerHTML =
        ↪  `<strong>Error:</strong> ${data.error}`;
```

```
            } else {
              const rawMarkdown = data.response || "No response
              ↪  received.";
              const htmlContent = marked.parse(rawMarkdown);
              responseElement.innerHTML =
              ↪  `<strong>Response:</strong><br>${htmlContent}`;
            }
          })
          .catch(error => {
            console.error("Error:", error);
            document.getElementById("response").innerHTML =
              `<strong>Error:</strong> Unable to connect to the
              ↪  chatbot.`;
          });
        })
        .catch(error => {
          console.error("Error fetching news:", error);
          document.getElementById("response").innerHTML =
            `<strong>Error:</strong> Unable to fetch news
            ↪  data.`;
        });
    }
  </script>
</body>
</html>
```