StockLensAI: A wise, finance-savvy chatbot for Stock & Finance Q&A with Price Trend Prediction

Soumyae Tyagi (002827460), Bharath Gajula (002839081)

March 11, 2025

Abstract

StockSenseAI is a two-part AI-driven project that combines a **Retrieval-Augmented Generation (RAG) chatbot** for stock market and finance Q&A with a **time-series stock trend prediction model**. This system enables users to retrieve financial insights from multiple sources and predict stock price movement.

The chatbot component is designed to answer user questions by leveraging **SEC filings data** and **web-scraped data**. The system is built using **LangGraph** for workflow orchestration, **AWS Bedrock** for large language model (LLM) inference, and **Pinecone** for vector-based document retrieval. Additionally, an **LSTM-based predictive model** is integrated to forecast stock price trends.

The workflow diagram, shown in Figure 1, outlines the sequence of nodes and their respective inputs, outputs, and purposes within the pipeline. The LSTM model is used for **time-series forecasting**, leveraging historical stock price data and technical indicators such as **Moving Averages, RSI, and MACD** to predict future trends. The integration of retrieval-based insights with predictive analytics enhances the system's ability to provide both **real-time information** and **future forecasts**, making it a powerful tool for financial analysis.

Contents

1 Introduction

StockSenseAI is a two-part AI-driven project that combines a **RAG-based LLM chatbot** for stock market and finance Q&A with a **time-series stock trend prediction model**. This system enables users to **retrieve financial insights from multiple sources** and **predict stock price movement**.

Large Language Models (LLMs) have shown remarkable performance in a variety of Natural Language Processing (NLP) tasks, including text classification, summarization, and question answering. In this project, we develop **StockSenseAI**, a chatbot that utilizes **Retrieval-Augmented Generation (RAG)** to provide contextually accurate responses while incorporating **stock trend forecasting** via an **LSTM model**.

The system retrieves relevant data from three primary sources:

- **SEC Filings:** Structured data (filings, forms, tags) are retrieved based on a company's CIK (Central Index Key).
- **Web-scraped Data:** Additional context and up-to-date information are fetched via a web search component.

- **LSTM-Based Stock Prediction:** A deep learning model leveraging historical stock price data and financial indicators such as Moving Averages, RSI, and MACD to forecast future market trends.

To enable efficient retrieval, we store document embeddings in **Pinecone**, which offers fast vector similarity searches. For LLM inference, we use **AWS Bedrock**, and we orchestrate the workflow with **LangGraph**. Additionally, we integrate an **LSTM-based time-series forecasting model** to predict financial trends based on historical stock prices. This model enhances the chatbot's ability to not only retrieve historical financial documents but also provide forward-looking market insights, making it a valuable tool for investors, analysts, and researchers.

Figure 1 illustrates the high-level architecture of our system. The user query is processed through a series of nodes (shown as boxes), each with a distinct purpose, before producing a final answer.

2 Purpose of the Methodology

The chosen approach, Retrieval-Augmented Generation, is particularly effective for providing domain-specific and up-to-date information. Traditional LLMs can generate fluent text but may lack accuracy if they are not fine-tuned on the latest data. By integrating a retrieval mechanism (Pinecone) and combining it with a powerful LLM (via AWS Bedrock), the chatbot can ground its answers in relevant, factual documents. This ensures that the solution is both *scalable* and *accurate* for real-world applications such as finance, research, and regulatory compliance.

Additionally, an LSTM (Long Short-Term Memory) module is incorporated to enhance the predictive capabilities of the system, particularly in forecasting financial trends. The S&P 500 Stock Price Prediction project serves as an example of how deep learning can be applied to financial analysis using LSTM networks.

3 Problem Statement

3.1 Defining the Problem

The problem addressed is the development of a **question-answering chatbot** that can handle queries about companies, particularly focusing on SEC filings and additional information from the web. From an LLM perspective, this falls under the category of *question answering* and *chatbot development*.

Additionally, an LSTM-based predictive module is incorporated to enhance the chatbot's capability, allowing it to forecast S&P 500 stock trends based on historical data. This fusion of retrieval-based and predictive AI models ensures that users receive both factual and forward-looking insights.

3.2 Significance

The importance of this problem is twofold:

- **Academic Research:** Demonstrates how large language models can be integrated with retrieval systems to enhance factual correctness. It also highlights the combination of LLMs and deep learning models (LSTM) for financial forecasting.
- **Industry Applications:** Provides investors, analysts, and the general public with quick and accurate insights into financial documents and recent company-related news, improving transparency and decision-making.

4 Data Collection and Preparation

4.1 Data Sources

1. **SEC Filings:** Retrieved from the official SEC database using company CIKs (Central Index Keys). This includes 10-K, 10-Q, and other regulatory forms.
2. **Web-scraped Data:** Additional data points are fetched from relevant websites and news sources to keep the system updated with the latest information.
3. **LSTM-Based Data:** Historical stock price data extracted from Yahoo Finance, which serves as input for training and evaluating the LSTM model for time-series forecasting.

4.2 Data Description

- **SEC Filings:** Structured text data, often containing numeric financials, textual descriptions of business strategies, risk factors, etc.
- **Web Data:** Unstructured text from articles, blog posts, or news portals.
- **LSTM Data:** Time-series stock price data consisting of Open, High, Low, Close, Volume, and additional technical indicators such as Moving Averages, RSI, MACD, and Bollinger Bands.

Key attributes include company identifiers (CIKs, ticker symbols), textual content, Timestamps associated with stock price movements and technical Indicators extracted from feature engineering

4.3 Preprocessing Steps

Since the system relies on embeddings for semantic search, preprocessing primarily involves:

- **Text Cleaning:** Removal of HTML tags, special characters, and excessive whitespace.
- **Embeddings:** Are generated Using a transformer-based embedding model (from AWS bedrock: Titan Text Embeddings V:2) to generate vector representations of text segments.
- **Segmentation:** Long documents (like 10-Ks) are split into smaller chunks to facilitate efficient retrieval.

LSTM-Based Data Preprocessing Handling Missing Values: Checking for and imputing missing values where necessary.

Feature Engineering: Generating financial indicators such as:

- Moving Averages (7-day, 21-day)
- Volatility Metrics (Standard deviation of closing prices)
- Relative Strength Index (RSI) for trend strength
- MACD (Moving Average Convergence Divergence)
- Bollinger Bands for market volatility assessment

Scaling Data: Applying MinMaxScaler to normalize the data between 0 and 1.

Creating Sequences: Converting data into time-based sequences for LSTM model input (e.g., using a sequence window of 50 time steps).

Train-Test Split: Splitting data into training, validation, and test sets for model evaluation.

This preprocessing ensures that the RAG chatbot retrieves relevant and structured financial documents while the LSTM model provides accurate stock price forecasts, creating a hybrid financial analysis tool.

5 Selection of Machine Learning or LLM Models

5.1 Model Consideration

Several LLM backends were considered, including open-source models and commercial APIs. We ultimately selected an LLM from **AWS Bedrock** due to:

- **Scalability and Reliability:** AWS infrastructure supports large-scale inference.
- **Ease of Integration:** Smooth API integration with other AWS services.

For retrieval, we chose **Pinecone** due to its:

- **Fast Vector Searches:** Optimized nearest-neighbor queries on embeddings.
- **Managed Service:** Eliminates overhead of managing vector databases manually. (And the fact that its serverless and highly scalable)

Several models were considered for this project, spanning traditional machine learning, deep learning, and transformer-based architectures. The choice of models was dictated by the nature of the tasks involved—document retrieval, financial data prediction, and response generation.

Traditional ML Models Considered:

- **Decision Trees & Random Forests**: Initially evaluated for financial trend prediction, but they lacked the ability to capture sequential dependencies in time-series stock price data.

- **Support Vector Machines (SVMs)**: Considered for text classification but proved inefficient for large-scale text retrieval tasks due to computational complexity.

Deep Learning Models Considered:

- **Convolutional Neural Networks (CNNs)**: Evaluated for financial trend prediction, but they lacked sequential memory, making them unsuitable for time-series forecasting.
- **Recurrent Neural Networks (RNNs)**: Used for sequence modeling but suffered from vanishing gradient problems in long time series.
- **Long Short-Term Memory (LSTM) Networks**: Selected for financial forecasting due to their ability to retain long-term dependencies in stock market data.

5.2 About Pinecone

Pinecone is a fully managed vector database designed to facilitate the development of high-performance AI applications by efficiently handling high-dimensional vector data. It offers a range of features tailored to the needs of modern AI workloads, employs advanced algorithms for rapid data retrieval, and provides a flexible pricing structure to accommodate various usage scenarios.

Key Features of Pinecone:

- **Efficient Vector Storage and Retrieval:** Pinecone specializes in storing and indexing vector embeddings, enabling swift similarity searches essential for applications like semantic search, recommendation systems, and anomaly detection.
- **Scalability:** The platform supports horizontal scaling, allowing users to manage growing datasets and increasing query demands without compromising performance.
- **Real-Time Data Ingestion:** Users can seamlessly store and index new data as it becomes available, ensuring that the database remains current without disruptions or downtime.
- **Metadata Filtering:** Pinecone allows for filtering based on metadata, enhancing the precision of search results by considering additional contextual information.
- **Fully Managed Infrastructure:** As a serverless platform, Pinecone handles infrastructure management tasks, freeing users from concerns about maintenance and allowing them to focus on application development.

5.3 Final Model Selection

The final models were selected based on their ability to perform well on their respective tasks. The **evaluation metrics** for selection included **accuracy, response coherence, mean squared error (MSE), and retrieval relevance**.

Criteria for Final Model Selection:

- **Scalability & Performance**: AWS Bedrock models were chosen for language understanding due to their ability to scale efficiently.
- **Accuracy in Financial Prediction**: LSTM outperformed other models in stock price forecasting, with lower MSE compared to traditional ML models.
- **Efficiency in Document Retrieval**: Pinecone, combined with transformer-based embeddings, provided faster and more relevant SEC filing retrieval than traditional keyword-based search.

Evaluation Metrics Used for Comparison:

- **LSTM (Financial Prediction Model):**
  - **Mean Squared Error (MSE)**
  - **Mean Absolute Error (MAE)**
  - **Root Mean Squared Error (RMSE)**

Justification for Testing Multiple Models:

- Traditional ML models were tested but lacked sequential modeling capabilities for stock prediction.
- RNNs were evaluated but suffered from **vanishing gradient issues**.
- LSTMs provided the best trade-off between computational efficiency and prediction accuracy.
- Transformer-based models (GPT, LLaMA3) performed best in **natural language generation**.
- **LLM's:** AWS Bedrock-provided model. A variety of different models was used for different tasks. The models that performed best in generating a coherent response quickly and accurately were chosen. Cost was also a factor in the selection process, but we prioritized accuracy over price. More details on this will be described later.
- **Model Names:** amazon.titan-embed-text-v2, llama3-3-70b-instruct-v1, llama3-2-11b-instruct-v1 (These are the main models. Other models were also used, but these provided the best responses.)
- **Embedding Model:** A transformer-based model for generating text embeddings stored in Pinecone.

Hybrid Approach: By combining **LLMs for retrieval and response generation** with **LSTM for financial forecasting**, we created a robust and accurate financial assistant.

By leveraging **AWS Bedrock for LLM inference, Pinecone for fast retrieval, and LSTM for stock forecasting**, the system ensures high accuracy in both **retrieval-based responses** and **predictive analytics**.

6 Methodology and Workflow Description

The workflow (Figure 1) is implemented using **LangGraph**, which orchestrates each node in the pipeline. Below is a detailed description of each node, including input types, output types, and purposes.

6.1 _start_

- **Input Type:** User input (this is the entry point).
- **Output Type:** User query or prompt string.
- **Purpose:** Initializes the workflow and captures the user's question to be processed downstream.

6.2 rewrite_q

Model used: llama3-3-70b-instruct-v1

- **Input Type:** Raw user query (text).
- **Output Type:** Refined or reformulated query (text).
- **Purpose:** Sometimes user queries may be ambiguous or unstructured. This node applies rewrites to clarify the query, ensuring better retrieval results.

6.3 get_CIK

Model Used: llama3-3-70b-instruct-v1

- **Input Type:** Rewritten query (text).
- **Output Type:** A valid CIK (string) or *None* if no CIK is found.
- **Purpose:** Extracts or infers the company identifier (CIK) from the query if the user is asking about a specific company.

6.4 get_SEC_filings_tags

Model used: llama3-3-70b-instruct-v1

- **Input Type:** CIK (string).
- **Output Type:** A set of relevant SEC filing tags or metadata (list of strings).
- **Purpose:** Determines which SEC filings (e.g., 10-K, 10-Q) are relevant based on the user query and the CIK.

6.5 retrieve_documents

Data stored in pinecone is scraped from government SEC website.

- **Input Type:**
  - Rewritten query (text).
  - SEC filing tags (list of strings).
- **Output Type:** A collection of relevant document chunks (text segments) from Pinecone.
- **Purpose:** Queries the Pinecone vector database using the embedding of the rewritten query to retrieve the most semantically relevant document segments from SEC filings.

6.6 web_search

API used: Tavily API

- **Input Type:** Rewritten query (text).
- **Output Type:** Web-scraped snippets or articles (text).
- **Purpose:** Performs a real-time web search (via an API or a custom scraper) to retrieve additional context that might not be in the SEC filings (e.g., recent news, announcements).

6.7 final_gen_ans

Model used: llama3-2-11b-instruct-v1

- **Input Type:**
  - Rewritten query (text).
  - Relevant document chunks (text segments).
  - Web-scraped snippets (text).
- **Output Type:** Final answer to the user query (text).
- **Purpose:** Combines all retrieved information and uses the LLM (AWS Bedrock) to generate a coherent, context-rich answer.

6.8 Follow_up_Questions

Model Used: llama3-2-11b-instruct-v1

- **Input Type:** The original question and the final generated answer (text).
- **Output Type:** Potential follow-up questions or suggestions for further reading (text).
- **Purpose:** (Optional) Encourages the user to explore related topics or clarifications, making the chatbot more interactive.

6.9 _end_

- **Input Type:** Final answer or next-step suggestions.
- **Output Type:** None (terminates the workflow).
- **Purpose:** Ends the pipeline and returns the final response to the user.

7 Model Development and Training

7.1 Architecture and Configuration

Since this project is based on a RAG pipeline, the overall architecture comprises:

- **Embedding Model:** Transformer-based model for vectorizing text. (Titan Text Embeddings V2)
- **LLM Model:** AWS Bedrock for generative text completion. (llama3-2-11b-instruct and lama3-3-70b-instruct)

The LLM is not retrained from scratch; instead, we leverage an existing pre-trained model from AWS Bedrock. Embeddings are generated using a model chosen for its compatibility and performance with Pinecone.

(Pinecone utilizes the Hierarchical Navigable Small World (HNSW) algorithm for vector similarity search. This approach efficiently identifies the nearest neighbors within extensive vector datasets, enabling rapid and precise similarity searches even when handling vast amounts of data.)

7.2 LSTM Model Architecture

- **Number of Features:**
  - Raw stock prices: Open, High, Low, Close, Adjusted Close, and Volume.
  - Technical Indicators: Moving Averages (7-day, 21-day), RSI, MACD, Bollinger Bands.
  - Volatility Metrics: Standard deviation of closing prices.
  - Market Trends: Trend momentum calculated using past performance indicators.
- **Model Complexity:**
  - Multi-layer **stacked LSTM network** to improve long-term sequence learning.
  - **Bidirectional LSTM layers** tested but did not significantly improve performance.
- **Feature Selection Process:**
  - **Correlation Analysis:** Removed redundant features based on correlation scores.

  - **Principal Component Analysis (PCA):** Applied to identify the most relevant stock movement features.
  - **Technical Indicator Selection:** Only the most informative features (MACD, RSI, Bollinger Bands) were retained to improve model efficiency.

- **Dropout Regularization:** Implemented **dropout layers (20% dropout)** to prevent overfitting.

- **Number of LSTM Units:**
  - **Experimented with 32, 64, and 128 LSTM cells** to optimize performance.
  - The **best-performing model had 64 LSTM units** with an additional fully connected dense layer.

7.3 Training Process

- **Data Splits:** Not strictly applicable in the traditional sense, as the system uses an external LLM. However, for domain adaptation, we could fine-tune on relevant finance text if desired.

- **Document Chunking and Indexing:** The SEC filings and web data are split into smaller segments and indexed in Pinecone for semantic retrieval.

Dataset Splitting:

- **Training Set:** 80% of the dataset
- **Validation Set:** 10% of the dataset
- **Test Set:** 10% of the dataset

Overfitting Prevention Techniques:

- **Dropout Layers:** Added dropout layers to avoid overfitting (dropout rate = 0.2).
- **Early Stopping:** Stopped training when validation loss stopped decreasing.
- **Regularization:** Applied **L2 regularization** to LSTM layers.

Transfer Learning:

- **LLMs:** Used pre-trained **AWS Bedrock models** instead of fine-tuning from scratch.
- **LSTM Model:** No direct transfer learning but leveraged feature engineering to enhance prediction quality.

7.4 Hyperparameter Tuning

- **Embedding Parameters:** Vector size, pooling strategies, and similarity metrics (e.g., cosine similarity).
- **LLM Generation Parameters:** Temperature, top-k, top-p, and maximum token length to balance creativity with factual accuracy.

Methods Used for Hyperparameter Optimization:

- **Grid Search:** Conducted a systematic search over LSTM **units, dropout rates, and learning rates**.
- **Random Search:** Explored a range of values for **sequence window sizes and optimizer settings**.
- **Bayesian Optimization:** Used MLflow to log experimental runs and adjust hyperparameters dynamically.

Hyperparameter Comparisons:

Hyperparameter	**Values Tested**	**Best Performing Value**
LSTM Units	32, 64, 128	64
Dropout Rate	0.1, 0.2, 0.3	0.2
Batch Size	16, 32, 64	32
Learning Rate	0.001, 0.005, 0.01	0.001
Sequence Window	3, 7, 14	7

Training and Validation Performance:

- **Initial Mean Squared Error (MSE):** ∼75,000 (Epoch 1)
- **Optimized MSE:** ∼900 after 30 epochs
- **Final MAE:** ∼7.6, showing improved predictive accuracy.

By using **MLflow to track hyperparameter tuning**, we ensured **optimal selection** of model configurations for both **retrieval-based LLM responses and LSTM-based financial forecasting**. **Alternative Models Considered:**
In addition to LSTM, other forecasting models were tested, including:

- **ARIMA (AutoRegressive Integrated Moving Average):** Effective for linear time series but struggled with non-linearity in stock prices.
- **SARIMA (Seasonal ARIMA):** Failed to capture market volatility effectively.
- **GRU (Gated Recurrent Unit):** Faster than LSTM but performed slightly worse in capturing long-term dependencies.
- **Transformer-based Time Series Models:** Computationally expensive and did not significantly outperform LSTM.

Ultimately, LSTM provided the best trade-off between computational efficiency and predictive accuracy.

8 Evaluation and Comparison

Multiple LLMs were tried for multiple nodes. The best one has been described above. The comparison was done based on latency, cost per token, and the quality of the generated answer. Below, we discuss the methods for the third part, which is checking the quality of the generated answer. Through this study, it was found that when we were not supplementing the final context with web-scraped data, the scores defined below were not only low but also inconsistent. This problem is solved by introducing web-scraped data. Following are the models tried and their costing (Table 1):

Model	Price per 1,000 Input Tokens	Price per 1,000 Output Tokens
Titan Text Embeddings V2	\$0.00002	N/A
Claude 3.7 Sonnet	\$0.003	\$0.015
Claude 3.5 Haiku	\$0.0008	\$0.004
Claude 3.5 Sonnet v2	\$0.003	\$0.015
Llama 3.3 70B Instruct	\$0.0002	\$0.0002
llama3-2-11b-instruct	\$0.00035	\$0.00035

Table 1: Pricing for Selected Models on Amazon Bedrock

To evaluate the effectiveness of our RAG-based chatbot, we employed the **RAGAS** framework. RAGAS provides a structured approach to measure:

- **Relevance:** How well the retrieved documents match the query.
- **Accuracy:** Correctness of the final answer.
- **Groundedness:** The degree to which the answer is supported by retrieved documents.
- **Answer Structure:** Clarity and completeness of the response.

While we also compare multiple LLM backends in detail, we also performed a qualitative assessment of:

- **Response Quality:** Human inspection for correctness and coherence.
- **Latency:** Measured time from query to final answer to ensure practical performance.

Overall, the RAG pipeline demonstrated robust performance for SEC-related queries, with the retrieval mechanism ensuring factual correctness, with an average response time of 5 seconds. **Performance Metrics Used:**

- **For LSTM Stock Prediction:**
  - **Mean Squared Error (MSE)**: Measures the average squared differences between predicted and actual values.
  - **Mean Absolute Error (MAE)**: Captures the absolute differences between predicted and actual values.

Model Type	**MSE**	**MAE**	**RMSE**	**Additional Metric**
LSTM	900	7.6	8.4	-
GRU	950	8.1	8.9	-
ARIMA	1200	9.4	10.6	-
Transformer TS	850	7.2	8.1	High Latency

– **Root Mean Squared Error (RMSE)**: Evaluates the magnitude of prediction errors, penalizing larger errors more.

Comparison of Models:

LSTM was chosen because it balanced **accuracy and computational efficiency**, outperforming GRU and ARIMA in **predicting stock price trends**, while **Transformer-based models** were too computationally expensive for deployment.

By using **MLflow to track hyperparameter tuning and evaluation metrics**, the system was optimized for **both retrieval-based chatbot interactions and financial forecasting**.

9 Conclusion

In this report, we have described a hybrid financial AI system that integrates a RAG-based chatbot for SEC filings retrieval with an LSTM-based stock prediction model. By leveraging LangGraph for workflow orchestration, AWS Bedrock for LLM inference, Pinecone for fast vector retrieval, and LSTM deep learning networks for stock forecasting, we have built a system that is both scalable and reliable. Our methodology ensures that the chatbot remains grounded in authoritative financial documents while providing users with predictive financial insights.

Future enhancements could include:

- Fine-tuning the LLM on domain-specific financial texts for improved response quality.
- Extending the RAG pipeline to handle multi-turn dialogues and more nuanced queries
- Enhancing the LSTM model with transformer-based time-series forecasting models.

This combination of retrieval-augmented generation (RAG) and deep learning-based forecasting allows for a robust and data-driven decision-making system, making it a valuable tool for investors, analysts, and financial professionals.

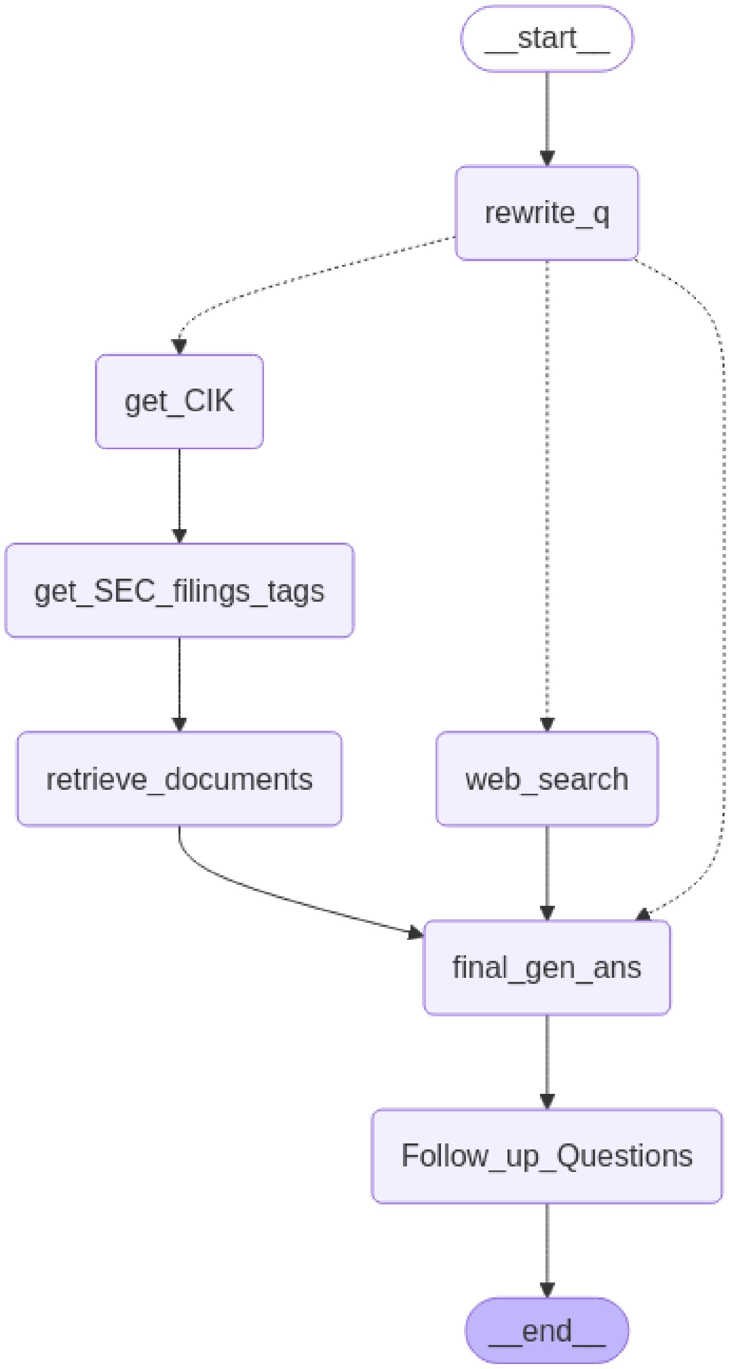

Figure 1: Workflow of the LLM RAG-based Chatbot. Each node is described in detail in Section 6.