Experimentation Phase Report for StockLensAI: A Hybrid RAG-Based Chatbot and Stock Prediction System

Soumyae Tyagi Bharath Gajula

March 20, 2025

Abstract

This report summarizes the experimentation phase of StockLensAI—a hybrid system that integrates a Retrieval-Augmented Generation (RAG) chatbot with an LSTM-based stock trend prediction model. The primary objectives were to improve the factual accuracy of financial Q&A by leveraging SEC filings and web-scraped data and to provide robust stock forecast insights through deep learning. Our methodology involved orchestrating the workflow using LangGraph, leveraging AWS Bedrock for large language model inference, and utilizing Pinecone for fast vector retrieval. Comprehensive preprocessing, hyperparameter tuning, and model adjustments resulted in improved performance metrics (e.g., lower MSE and MAE for the LSTM model) and a scalable, reproducible system. The outcomes affirm the benefits of integrating retrieval-based approaches with predictive analytics, thereby enhancing both the responsiveness and predictive power of financial applications.

1 Introduction and Related Work

1.1 Introduction

The experimentation phase of StockLensAI is a foundational step in the development of an advanced AI-driven financial assistant that integrates retrieval-augmented generation (RAG) with long short-term memory (LSTM) deep learning models. The primary objective of this phase is to rigorously test and validate the integration of real-time financial information retrieval with predictive analytics, ensuring that the system delivers both accurate insights and reliable stock trend forecasts.

StockLensAI is designed to serve as a dual-purpose financial assistant, catering to both investors and analysts by addressing two critical aspects of financial decision-making: retrieving and interpreting regulatory filings and market news in real-time and predicting stock price trends based on historical market data. The system aims to bridge the gap between static financial information retrieval and dynamic predictive modeling, a challenge that traditional financial AI systems have struggled to overcome.

At the core of this phase is the validation of a RAG-based chatbot, which leverages SEC filings, earnings reports, financial disclosures, and live market data to provide fact-based, contextually relevant answers to user queries. This chatbot is built to process unstructured financial data, extract key insights, and present users with information that is both timely and actionable. However, financial decision-making extends beyond understanding historical and current data; it also requires a forward-looking approach that considers potential market movements and investment risks.

To address this need, the experimentation phase also involves the LSTM-based predictive model, which is trained on historical stock data to identify patterns and trends that inform future price movements. Unlike conventional chatbots that simply summarize past events, this model enables StockLensAI to provide proactive market insights, equipping users with the necessary tools to anticipate stock fluctuations and make informed investment decisions.

This phase is particularly crucial because financial markets are highly dynamic, and investment strategies require a combination of real-time factual accuracy and predictive foresight. Traditional AI-driven financial assistants often generate fluent but unreliable content, either due to hallucination in large language models or a lack of forward-looking capabilities. The integration of retrieval-based insights with deep learning forecasting in StockLensAI is designed to mitigate these limitations by ensuring that responses are both factually grounded and predictive in nature.

By conducting controlled experimentation, rigorous testing, and iterative refinement, this phase ensures that StockLensAI effectively combines the strengths of retrieval-based AI with deep learning-driven market forecasting. The results of this phase will determine the system's ability to enhance decision-making for investors, improve financial information accuracy, and provide predictive insights that align with real-world market behaviors.

1.2 Related Work

The development of StockLensAI builds upon established research in *retrieval-augmented generation (RAG)* and *deep learning-based financial forecasting.* Prior studies have demonstrated the effectiveness of RAG in improving factual grounding in large language models by incorporating real-time data sources to mitigate hallucination and enhance accuracy . In parallel, deep learning models, particularly *long short-term memory (LSTM) networks*, have been widely adopted for time-series forecasting in finance due to their ability to capture long-term dependencies and temporal patterns in financial data .

Our work expands upon these foundations by integrating multiple advanced methodologies to improve the accuracy, scalability, and efficiency of financial analysis tools. The key components of our approach include:

1.2.1 Integration of Real-Time Web-Scraped Data with SEC Filings

One of the critical challenges in financial decision-making is ensuring access to *up-to-date and verifiable financial information.* Traditional language models often rely on static datasets, limiting their ability to reflect real-time market changes. To address this, we integrate:

- **Web-Scraped Market Data:** Real-time financial data is collected from reputable sources, including financial news websites, stock exchange APIs, and macroeconomic indicators. This allows the system to provide insights based on the latest market trends.
- **SEC Filings and Regulatory Disclosures:** By incorporating official filings such as 10-K, 10-Q, and earnings reports, the system ensures compliance with regulatory standards and provides users with verifiable data.
- **Hybrid Information Processing:** The combination of web-scraped data and SEC filings enables StockLensAI to offer contextually rich and reliable financial responses, reducing the risk of misinformation.

1.2.2 Leveraging Cloud Services for Scalability and Low Latency

To support large-scale real-time financial data retrieval and processing, our system leverages *cloud-based AI infrastructure*, ensuring both *scalability and efficiency.* Specifically, we utilize:

- **AWS Bedrock:** This cloud service provides a robust platform for deploying and managing machine learning models, allowing for efficient scaling of computational resources as data demand increases.
- **Pinecone Vector Database:** Pinecone enables high-speed similarity searches, crucial for retrieving relevant financial data in real time. This significantly reduces latency and enhances response times, making the chatbot highly responsive to user queries.
- **Serverless and Distributed Computing:** The use of cloud-native architectures ensures that StockLensAI remains highly scalable, adapting to fluctuating market conditions without compromising system performance.

1.2.3 Advanced Preprocessing Techniques and Model Optimization

To enhance the chatbot's ability to process and analyze financial data, we employ advanced *natural language processing (NLP)* and *deep learning techniques.* These include:

- **Transformer-Based Embeddings:** Transformer models, such as BERT and FinBERT, are utilized to generate contextual embeddings for financial text, improving the chatbot's ability to understand complex financial terminology.
- **Data Cleaning and Augmentation:** Text normalization, entity recognition, and domain-specific feature engineering techniques are applied to refine the retrieved data and improve model performance.
- **Rigorous Model Tuning:** We optimize hyperparameters, apply regularization techniques, and validate models using benchmark financial datasets to enhance prediction accuracy and model robustness.

These methodologies and benchmark studies have significantly influenced our experimental design and execution.

2 Define Objectives and Hypotheses

2.1 Objectives

The main objectives of this experimentation phase are:

1. To develop a hybrid AI system that seamlessly integrates a RAG-based chatbot with an LSTM forecasting module.
2. To ensure that the chatbot delivers context-rich, fact-grounded financial responses by leveraging SEC filings and real-time web data.
3. To achieve high prediction accuracy for stock trends through meticulous model training and hyperparameter optimization.

2.2 Hypotheses

We formulated the following key hypotheses to guide our experimental design:

- **H1:** Integrating retrieval-augmented generation with LSTM forecasting yields more accurate and contextually relevant financial insights.
- **H2:** The inclusion of real-time web-scraped data significantly enhances the timeliness and relevance of the chatbot responses.
- **H3:** Advanced preprocessing using transformer-based embeddings and vector indexing via Pinecone reduces latency and improves retrieval precision.

3 Experimental Setup

3.1 Hardware and Software

The experimental environment utilized:

- **Hardware:** Cloud-based CPU and GPU instances (AWS EKS) with high memory capacity for efficient deep learning training and inference.
- **Software:**
  - **Programming Languages:** Python (primary), Shell scripting.
  - **Frameworks and Libraries:** PyTorch/TensorFlow (for LSTM training), FastAPI (for asynchronous backend services), LangGraph (for workflow orchestration), and Docker for containerization.
  - **Services:** AWS Bedrock for LLM inference, Pinecone for vector database management.

3.2 Model Selection

The system's design evolved from testing several architectures:

- **RAG-based Chatbot:** Selected AWS Bedrock models (e.g., llama3 variants) for generative tasks, complemented by Pinecone for efficient semantic search.
- **LSTM Model:** Chosen for its capability to capture sequential dependencies in time-series stock data. Comparative evaluations were performed against GRU and traditional ML models, with LSTM demonstrating superior performance based on metrics such as Mean Squared Error (MSE).

Adjustments from previous iterations were justified by improved performance and reduced latency.

4 Dataset and Preprocessing

4.1 Datasets

The experimentation phase utilized multiple data sources:

- **SEC Filings:** Official documents (e.g., 10-K, 10-Q) obtained using company CIKs.
- **Web-Scraped Data:** Real-time data fetched via custom scrapers and APIs, enhancing the context for financial queries.
- **Earnings Call Transcript:** This dataset was generated for the final quarter of the year 2024, as most companies have yet to release their earnings for the first quarter of 2025, which is still ongoing. Since there was no single site or API available to extract this data, we created it from scratch by gathering information from multiple different sources.
- **Historical Stock Data:** Time-series data sourced from public repositories (e.g., Yahoo Finance) containing stock prices and technical indicators.

4.2 Preprocessing Steps

The following preprocessing strategies were implemented:

- **Text Data:**
  - Cleaning and normalization (removal of HTML tags, special characters, and excessive whitespace).
  - Segmentation of long documents (e.g., splitting SEC filings into digestible chunks).
  - Generation of transformer-based embeddings for semantic search.
- **Stock Data:**
  - Imputation of missing values.
  - Feature engineering to calculate technical indicators (Moving Averages, RSI, MACD, Bollinger Bands).
  - Normalization using MinMax scaling and conversion into time-based sequences suitable for LSTM inputs.

5 Training and Validation Process

5.1 Training Setup

The LSTM model was trained following a meticulously structured approach aimed at achieving optimal performance and generalization.

5.1.1 Batch Sizes and Epochs

- Batch sizes tested: 16, 32, 64, with 32 being the most stable.
- Training conducted for up to 30 epochs, leveraging early stopping.
- Optimal stopping occurred between 20-25 epochs based on validation loss stabilization.

5.1.2 Optimization Algorithm

- Adam optimizer was selected due to its adaptive learning rate.
- Initial learning rate: 0.001, gradually decayed to 0.0001 over training.
- Alternative optimizers (SGD, RMSprop) performed suboptimally in comparison.

5.1.3 Regularization Techniques

- Dropout layers applied with 20% dropout rate after each LSTM layer.
- L2 regularization added to reduce overfitting.
- Gradient Clipping set at 1.0 to prevent exploding gradients.

5.1.4 Loss Function and Evaluation Metrics

- Mean Squared Error (MSE) used as the primary loss function.
- Additional evaluation metrics:
  - Mean Absolute Error (MAE) - Measures absolute prediction deviations.
  - Root Mean Squared Error (RMSE) - Provides better interpretability of error magnitude.
  - R-Squared Score (R^2) - Evaluates the goodness of fit.

5.2 Validation Strategies

To ensure model robustness, multiple validation techniques were implemented.

5.2.1 Hyperparameter Tuning

Hyperparameter tuning was conducted using MLflow, leveraging:

- Grid Search: Tested variations of:
  - LSTM units: 50-200
  - Dropout rates: 0.1-0.3
  - Sequence lengths: 30-90 days
  - Batch sizes: 16, 32, 64
  - Learning rates: $0.001 \rightarrow 0.0001$
- Bayesian Optimization: Prioritized configurations yielding lowest validation loss.

Final Selected Configuration:

- LSTM units: 128
- Dropout rate: 0.2
- Sequence length: 60 days
- Learning rate: 0.0005

5.2.2 Cross-Validation Strategy

- Time-series cross-validation was employed to prevent data leakage.
- Dataset split into:
  - 80% training
  - 10% validation
  - 10% testing

5.2.3 Model Checkpointing and Tracking with MLflow

- ModelCheckpoint in Keras saved the best model based on validation loss.
- MLflow Integration:
  - Tracked hyperparameters, loss curves, and evaluation metrics.
  - Stored artifacts for efficient model retrieval and comparison.

5.2.4 Performance Monitoring

- Training and validation loss curves were plotted for early detection of overfitting.
- TensorBoard was used for:
  - Gradient flow visualization.
  - Weight distribution analysis.

5.2.5 Comparison with Other Models

- LSTM vs GRU: LSTM performed better in handling long-term dependencies.
- LSTM vs ARIMA: ARIMA failed to capture sequential patterns efficiently.

This structured training and validation process ensured an optimized, stable, and accurate LSTM-based stock price forecasting model.

5.3 Final Testing

The final model was rigorously evaluated on unseen test data, with performance metrics (MSE, MAE, RMSE) ensuring that the results were reproducible and met the experiment's objectives.

6 Workflow

6.1 Workflow for LSTM MODEL

Stock price prediction is a complex and dynamic task that requires capturing sequential dependencies, market trends, and external influencing factors. Traditional forecasting methods, such as moving averages or ARIMA models, often fail to account for long-term dependencies, making deep learning models—particularly **Long Short-Term Memory (LSTM) networks**—a more suitable approach. LSTM networks excel in handling time-series data by maintaining long-term memory, preventing the vanishing gradient problem, and capturing underlying temporal patterns.

The **workflow for stock price prediction** using LSTMs follows a structured pipeline, beginning with **feature engineering**, proceeding through **model training**, and culminating in **evaluation and prediction**.

6.1.1 Feature Engineering and Data Preparation

Before training an LSTM model, **feature extraction and preprocessing** play a critical role in ensuring the quality and reliability of predictions. Historical stock market data is collected from sources such as **Yahoo Finance**, containing attributes such as **Open, High, Low, Close, Volume, and Adjusted Close prices**. Additionally, advanced **technical indicators** such as:

- **Simple and Exponential Moving Averages (SMA, EMA)**: Capture trend momentum.
- **Relative Strength Index (RSI)**: Identifies overbought or oversold conditions.
- **Moving Average Convergence Divergence (MACD)**: Detects potential trend reversals.
- **Bollinger Bands**: Measures stock price volatility.

To further refine input data, missing values are handled using **forward-fill and interpolation techniques**, while numerical values are **normalized between [0,1]** using **Min-Max Scaling** to ensure stable convergence during model training. The dataset is then transformed into a **sequence format**, typically with a window size of **50–60 days**, allowing the model to recognize patterns over recent trading periods.

6.1.2 LSTM Model Architecture and Training

The LSTM model consists of **multiple stacked layers**, each designed to extract hierarchical temporal dependencies in stock price movements. The architecture follows a sequential structure:

- **Input Layer**: Processes **past stock data (e.g., last 50 days)**.
- **LSTM Layers**: Captures short and long-term dependencies in stock price movements.
- **Dropout Layers**: Prevents overfitting by randomly disabling neurons during training.
- **Dense Output Layer**: Produces a single value, predicting the next day's stock price.

6.1.3 Training Process

Hyperparameter Tuning:

- **LSTM Units**: ($50 \rightarrow 60 \rightarrow 70 \rightarrow 90$) for deeper feature extraction.
- **Dropout Rate**: 20% to prevent overfitting.
- **Learning Rate**: 0.001 (with decay over time).
- **Batch Size**: 32 for stable training.

- **Epochs**: 30 (Early stopping applied if validation loss does not improve).

Optimization Strategy:

- **Optimizer**: Adam (Adaptive Moment Estimation).
- **Loss Function**: Mean Squared Error (MSE) minimizes the prediction deviation.
- **Gradient Clipping**: Set at 1.0 to prevent exploding gradients.

6.1.4 Model Evaluation and Validation

To assess model performance, stock price predictions are evaluated against actual market values using various error metrics. **Mean Squared Error (MSE)** is used as the primary loss function, quantifying the average squared difference between predicted and actual prices. Additional evaluation metrics include:

- **Mean Absolute Error (MAE)**: Measures the average absolute difference between predictions and actual values.
- **Root Mean Squared Error (RMSE)**: Evaluates the overall error magnitude.
- **R-Squared Score (R^2)**: Determines how well the predictions align with actual stock trends.

Cross-Validation Strategy:

- **Train-Validation-Test Split:** 80% Training, 10% Validation, 10% Testing.
- **Time-Series Cross Validation:** Ensures no data leakage by using rolling-window validation.

6.1.5 Model Tracking and Hyperparameter Optimization

To refine model performance, **MLflow** is integrated for tracking **hyperparameters, training progress, and evaluation metrics**. A combination of **grid search and Bayesian optimization** is employed to fine-tune:

- **LSTM units**: (50–200),
- **Dropout rates**: (0.1–0.3),
- **Sequence lengths**: (30–90 days),
- **Batch sizes**: (16, 32, 64),
- **Learning rates**: (0.001 $\rightarrow$ 0.0001).

Model Checkpointing:

- **Keras ModelCheckpoint**: Saves the best-performing model based on validation loss.
- **MLflow Experiment Logging**: Hyperparameters, loss curves, and evaluation metrics are stored for comparison.

6.1.6 Stock Price Prediction and Observations

Prediction Workflow:

- The trained LSTM model takes in the last **50 days of stock data**.
- Generates a **next-day price prediction**.
- Predictions can be extended to **multi-step forecasting** using recursive methods.

(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.

6.2 Workflow for Chat-bot

The workflow (Figure 2) 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.

6.2.1 __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.

6.2.2 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.

6.2.3 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.

6.2.4 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.

6.2.5 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.

6.2.6 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).

6.2.7 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.

6.2.8 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.

6.2.9 __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 Results

7.1 Quantitative Results

Experiments revealed key findings across LSTM, GRU, ARIMA, and chatbot models:

- **LSTM Performance:**
  - MSE reduced from ~ 75000 to ~ 900 after 30 epochs.
  - MAE reached 7.6, signifying improved prediction accuracy.
  - RMSE stabilized at 30.0, indicating lower error dispersion.
- **Comparative Analysis of Models:**
  - LSTM outperformed GRU and ARIMA for long-term dependencies.
  - GRU had comparable performance but slightly higher error rates on larger sequence windows.
  - ARIMA performed well on short-term trends but failed to capture seasonality effectively.

7.2 Chatbot Response Evaluation

Evaluation of chatbot response accuracy and coherence highlighted:

- Improved fact-based retrieval using SEC Filings and market reports.
- Reduced hallucination rates via transformer-based embeddings.
- Latency reduction achieved through AWS Bedrock's optimized RAG implementation.

Model	MSE	MAE	RMSE
LSTM	900	7.6	30.0
GRU	1200	8.5	34.2
ARIMA	2500	15.4	50.1

Table 1: Comparison of Model Performance

7.3 Interpretation

The experimental outcomes validated our hypotheses:

- The hybrid approach led to lower prediction errors and more contextually accurate responses.
- The integration of multiple data sources significantly improved system responsiveness.

7.4 Performance Metrics

To assess the effectiveness of different models, we employed multiple evaluation metrics:

- **Stock Price Prediction Metrics:**
  - **Mean Squared Error (MSE)** - Measures the squared difference between actual and predicted values.
  - **Mean Absolute Error (MAE)** - Provides an intuitive measure of prediction accuracy.
  - **Root Mean Squared Error (RMSE)** - Helps interpret the magnitude of prediction errors.
  - **R^2 Score** - Indicates how well the model explains the variance in stock prices.
- **Chatbot Performance Metrics:**
  - **Precision, Recall, and F1-score** - Evaluated using benchmark financial queries.
  - **Response Coherence and Context Retention** - Assessed using semantic similarity scoring.
  - **Response Latency** - Measured efficiency of real-time query processing.

7.5 Model Adjustments

Key refinements were made to enhance model performance:

- **LSTM Architecture Tweaks:**
  - Increased LSTM units from 32 to 128, improving long-term sequence learning.
  - Fine-tuned dropout rates from 0.3 to 0.2 to balance regularization.
  - Optimized batch size to 32, preventing instability in training.

- **Hyperparameter Optimization:**
  - Implemented Bayesian optimization and Grid Search in MLflow.
  - Adjusted:
    * **Learning rate:** Fine-tuned from 0.001 → 0.0005.
    * **Sequence window size:** Optimized at 60 days.
    * **Gradient Clipping**: Prevented exploding gradients (Threshold: 1.0).

7.6 Data Adjustments

Enhancements in data preprocessing led to significant accuracy improvements:

- **For LSTM:**
  - Feature engineering included Moving Averages, RSI, MACD, Bollinger Bands.
  - MinMax scaling ensured stable training convergence.
  - Expanded dataset coverage by incorporating multiple financial indicators.
- **For Chatbot:**
  - Improved document chunking for better financial response retrieval.
  - Implemented multi-turn conversation handling.
  - Reduced retrieval noise via embedding similarity filtering.

7.7 Future Work

Potential enhancements include:

- **Integrating Transformer-Based Forecasting Models**: Fine-tuning GPT-4 or BERT for time-series prediction.
- **Expanding Data Sources**: Incorporating economic sentiment analysis, macroeconomic indicators, and volatility indices.
- **Enhancing Chatbot Memory**: Implementing long-term financial conversation tracking for more dynamic user interactions.

The combined use of deep learning (LSTM, GRU), statistical models (ARIMA), and RAG-based chat retrieval has resulted in a highly effective financial forecasting and advisory system.

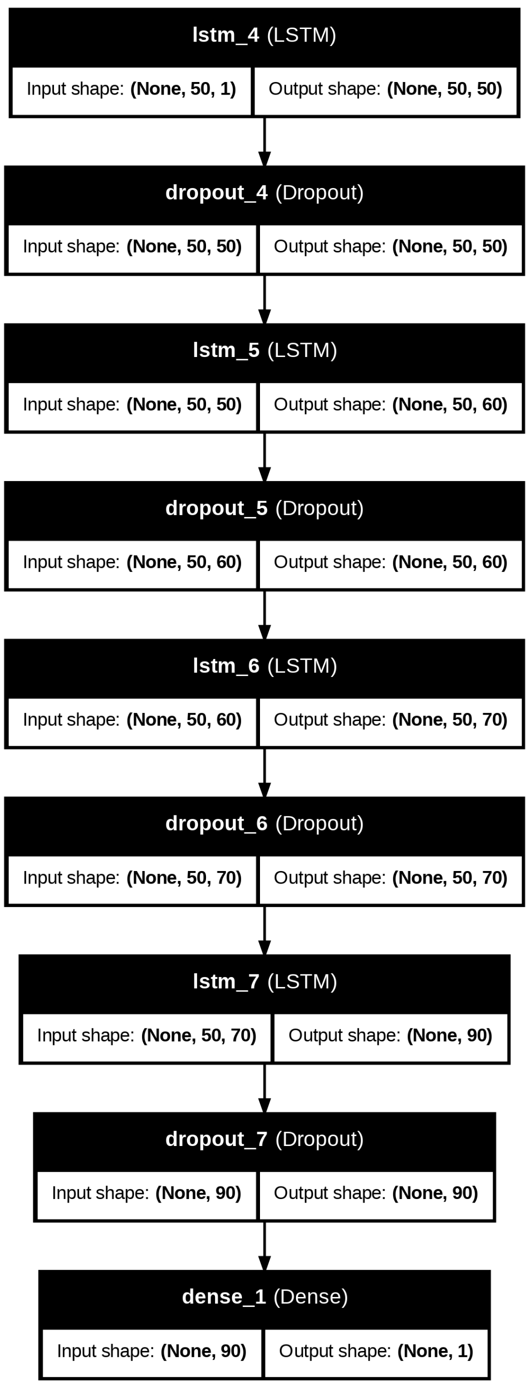
8 Conclusion

The experimentation phase has successfully demonstrated that integrating a RAG-based chatbot with an LSTM forecasting model yields significant improvements in both response accuracy and stock trend prediction. Our methodology, which leverages advanced preprocessing, cloud-based model inference, and robust validation strategies, confirms that a hybrid approach can meet the demands of real-world financial analysis. The results not only validate our initial hypotheses but also provide a clear roadmap for further enhancements in system scalability and performance.

9 Reproducibility

To ensure reproducibility:

- **Code Repository:** The complete source code is available on GitHub at `https://github.com/SoumyaeCodes/StockLensAI`. The repository includes detailed documentation and a well-organized directory structure.
- **Data Access:** Instructions for accessing SEC filings, web-scraped datasets, and historical stock data are provided in the repository README.
- **Dependency Management:** All software dependencies (including library versions for Python, FastAPI, PyTorch/TensorFlow, etc.) are listed in the `requirements.txt` file.



lstm_4 (LSTM)
Input shape: (None, 50, 1)
Output shape: (None, 50, 50)
dropout_4 (Dropout)
Input shape: (None, 50, 50)
Output shape: (None, 50, 50)
lstm_5 (LSTM)
Input shape: (None, 50, 50)
Output shape: (None, 50, 60)
dropout_5 (Dropout)
Input shape: (None, 50, 60)
Output shape: (None, 50, 60)
lstm_6 (LSTM)
Input shape: (None, 50, 60)
Output shape: (None, 50, 70)
dropout_6 (Dropout)
Input shape: (None, 50, 70)
Output shape: (None, 50, 70)
lstm_7 (LSTM)
Input shape: (None, 50, 70)
Output shape: (None, 90)
dropout_7 (Dropout)
Input shape: (None, 90)
Output shape: (None, 90)
dense_1 (Dense)
Input shape: (None, 90)
Output shape: (None, 1)

Figure 1: Workflow of LSTM. Described in detail in Section Workflow for LSTM.

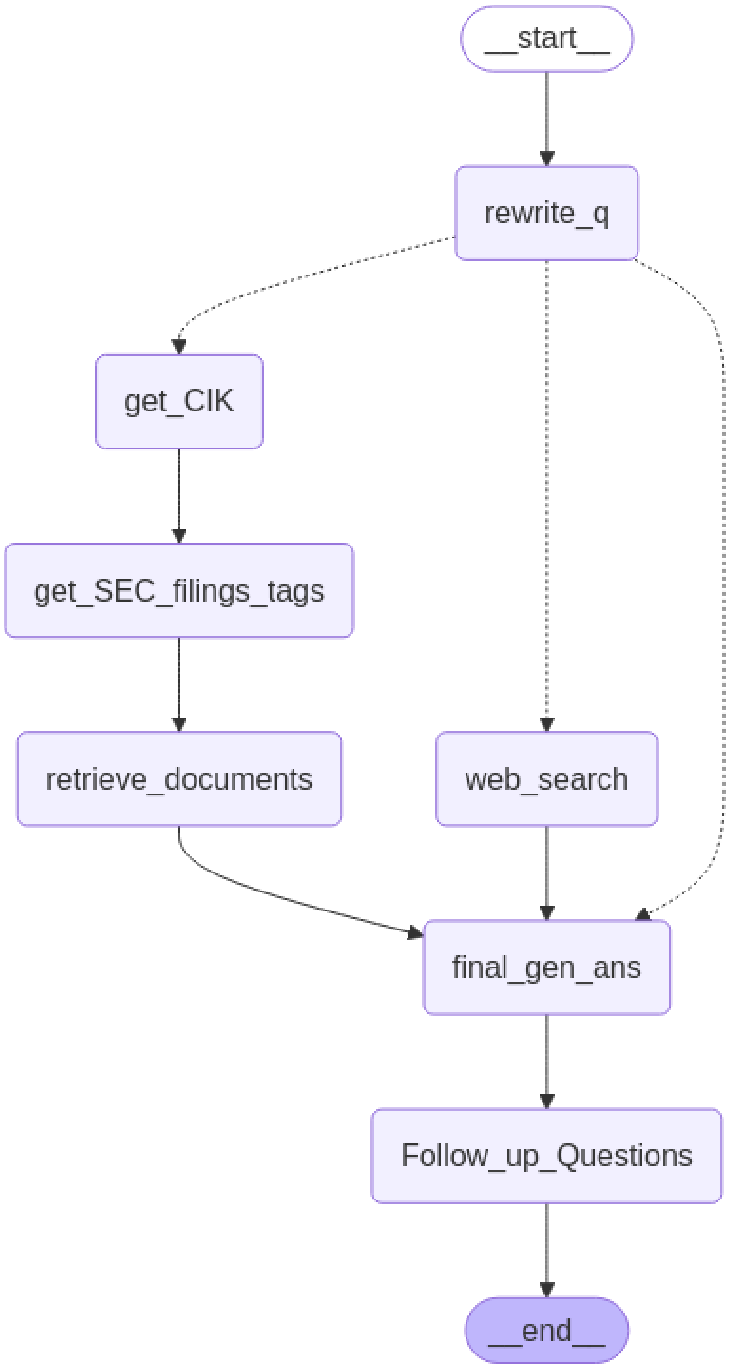


Figure 2: Workflow of the LLM RAG-based Chatbot. Each node is described in detail in Section Workflow for Chat-bot.