Subtask 4 : Implementation and Evaluation of the Model's Performance

Aman Rajput[†] 202418003@daiict.ac.in
Adarsh Ambastha[†] 202418004@daiict.ac.in
Mihir Pandya[§] 202418032@daiict.ac.in
Vaibhav Agrawal[†] 202418059@daiict.ac.in

Abstract

This project presents a sentiment-based time series forecasting approach applied to Amazon reviews within the "Baby Products" category. The initial dataset, provided in JSONL format, was ingested and preprocessed—converting it to CSV format and standardizing timestamp fields. Verified purchases were filtered to ensure data reliability, and products were grouped by `parent_asin`, with the second-most reviewed product selected for further analysis. Due to the absence of labeled sentiment data, we employed a pretrained RoBERTa transformer model to assign sentiment scores (Positive, Neutral, Negative) using a softmax probability distribution. A manually labeled subset of 300 reviews was used to validate the sentiment model, achieving an F1 score of 0.72 and accuracy of 67%. Fine-tuning the RoBERTa model with this subset improved the performance to an F1 score of 0.75 and accuracy of 82%, highlighting the benefits of even minimal supervised learning. Sentiment scores were then forecasted over time using four models: ARIMA, LSTM, Stacked LSTM, and KerasTuner-optimized Stacked LSTM. Evaluation metrics—RMSE, MSE, MAE, and R^2—demonstrated that the KerasTuner-optimized Stacked LSTM consistently outperformed baseline models across all sentiment classes. This pipeline showcases a robust method for sentiment-aware time series forecasting using a combination of transformer-based NLP and deep learning architectures.

End-to-End Architecture of the Model

Pipeline

1. Data Ingestion Preprocessing The raw data was provided in `.jsonl` (JSON Lines) format. Using Python's `json` and `pandas` libraries, each line was parsed and converted into a structured tabular format, then saved as a `.csv` file. The `timestamp` column was converted into a standardized datetime object to enable chronological ordering and temporal analysis.

2. Filtering Grouping Next, the dataset was filtered to retain only verified purchases (`verified_purchase=True`) to ensure data authenticity. The reviews were then grouped using the `parent_asin` identifier. The product with the second-highest number of reviews was selected for detailed analysis. This subset was saved separately under the name `TopProduct_2.csv`.

3. Sentiment Scoring with RoBERTa and Time Series Forecasting

- **Input**: Text reviews
  - Long reviews → `facebook/bart-large-cnn` summarizer
  - Short reviews → Direct processing
- **Sentiment Analysis**:cardiffnlp/twitter-roberta-base-sentiment
  - Outputs: `neg`, `neu`, `pos` scores
- **Forecasting**: Time-series → Stacked LSTM with KerasTuner (Table 3)

Implementation Details

The implementation of the project followed a structured pipeline, starting with the ingestion and preprocessing of Amazon reviews in the Baby Products category. The raw data was provided in `.jsonl` (JSON Lines) format, which was converted into a more accessible `.csv` format using Python's `pandas` and `json` libraries. The `timestamp` column was standardized into a proper datetime format using `pandas.to_datetime()` for time series compatibility. Reviews were then filtered to include only those marked as verified purchases. For product-level analysis, reviews were grouped based on the `parent_asin` identifier, and the product with the second-highest number of reviews was selected for focused study—this subset was saved as `TopProduct_2.csv`.

To assess the sentiment of reviews, a pretrained RoBERTa model (`cardiffnlp/twitter-roberta-base-sentiment`) was used. Each review was tokenized and passed through the model to obtain softmax-based probability scores for positive, neutral, and negative sentiment classes. Since the dataset lacked ground-truth sentiment labels, we manually annotated approximately 300 reviews and used these for

both validation and fine-tuning of the sentiment model. When compared against manual labels, the base model achieved an F1 score of 0.72, precision of 0.86, recall of 0.67, and accuracy of 0.67. After fine-tuning RoBERTa on an 80-20 train-test split of the labeled data, the model showed improved performance with an F1 score of 0.75, precision of 0.81, recall of 0.82, and accuracy of 0.82. This fine-tuned model was then used to assign sentiment scores to the entire dataset.

Following sentiment scoring, daily average sentiment values for each class (positive, neutral, and negative) were computed to form three univariate time series. Four forecasting models—ARIMA, LSTM (Table 1), Stacked LSTM (Table 2), and Stacked LSTM with KerasTuner (Table 3)—were implemented to predict future sentiment trends. ARIMA modeling was carried out using the `statsmodels` library. LSTM models were built using TensorFlow/Keras, where a simple LSTM consisted of a single hidden layer with 64 units, while the stacked variant included three LSTM layers. The KerasTuner framework was used to optimize hyperparameters such as the number of units, learning rate, and batch size. Finally, all models were evaluated using standard metrics—RMSE, MAE, MSE, and R^2—demonstrating that the KerasTuner-optimized Stacked LSTM achieved the best performance across 2 out of 3 sentiment categories.

Table 1: Summary of the LSTM Model Architecture

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
dense (Dense)	(None, 32)	2,080
dense_1 (Dense)	(None, 1)	33
Total params: 18,009 (74.25 KB)		
Trainable params: 18,009 (74.25 KB)		
Non-trainable params: 0 (0.00 B)		

Table 2: Summary of the Stacked LSTM Model Architecture

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 30, 64)	16,896
dropout_6 (Dropout)	(None, 30, 64)	0
lstm_7 (LSTM)	(None, 30, 64)	33,024
dropout_7 (Dropout)	(None, 30, 64)	0
lstm_8 (LSTM)	(None, 64)	33,024
dropout_8 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 8)	520
dense_5 (Dense)	(None, 1)	9
Total params: 83,473 (326.07 KB)		
Trainable params: 83,473 (326.07 KB)		
Non-trainable params: 0 (0.00 B)		

Table 3: Summary of the KerasTuner-Optimized Stacked LSTM Model Architecture

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 30, 128)	66,560
dropout (Dropout)	(None, 30, 128)	0
lstm_1 (LSTM)	(None, 96)	86,400
dropout_1 (Dropout)	(None, 96)	0
dense (Dense)	(None, 12)	1164
dense_1 (Dense)	(None, 1)	13
Total params: 154,137 (602.10 KB)		
Trainable params: 154,137 (602.10 KB)		
Non-trainable params: 0 (0.00 B)		

Baselines

Neutral	RMSE	MSE	MAE	R^2
ARIMA	0.1228	0.0151	0.1188	-23.05
LSTM	0.0266	0.0007	0.0204	0.8442
S-LSTM	0.0072✓	0.0001	0.0058✓	0.9167✓
S-LSTM(KT)	0.0079	0.00006✓	0.0062	0.8667

Table 4: Performance of Models on Neutral Class

Positive	RMSE	MSE	MAE	R^2
ARIMA	0.1001	0.0100	0.0841	-2.771
LSTM	0.0536	0.0029	0.0436	0.8534
S-LSTM	0.0148	0.00022	0.0119	0.9049
S-LSTM(KT)	0.0132✓	0.00017✓	0.0102✓	0.9168✓

Table 5: Performance of Models on Positive Class

Negative	RMSE	MSE	MAE	R^2
ARIMA	0.1081	0.0117	0.1038	-13.28
LSTM	0.0526	0.0028	0.0426	0.8680
S-LSTM	0.0085	0.0001	0.0069	0.9113
S-LSTM(KT)	0.0062✓	0.00003✓	0.0047✓	0.9429✓

Table 6: Performance of Models on Negative Class

1. ARIMA vs KerasTuner-Optimized Stacked LSTM Model Architecture

The ARIMA model, a traditional statistical approach, consistently underperforms compared to the KerasTuner-Optimized Stacked LSTM across all sentiment classes.
For the **Neutral class** (Table 4), ARIMA exhibits a high RMSE of 0.1228 and MSE of 0.0151, with a significantly negative R^2 value of -23.05, indicating that it fails to capture the underlying patterns in the data. In contrast, the KerasTuner-Optimized Stacked LSTM achieves a much

lower RMSE of 0.0079, MSE of 0.00006, and a positive R^2 of 0.867, demonstrating a strong fit to the data.

For the **Positive class** (Table 5), ARIMA's RMSE is 0.1001, MSE is 0.0100, and R^2 is -2.771, again reflecting poor predictive capability. The KerasTuner-Optimized Stacked LSTM, however, reduces the RMSE to 0.0132, MSE to 0.00017, and achieves an R^2 of 0.9168, indicating superior performance.

The **Negative class** (Table 6) follows a similar trend, with ARIMA's RMSE at 0.1081, MSE at 0.0117, and R^2 at -13.28, while the KerasTuner-Optimized Stacked LSTM achieves an RMSE of 0.0062, MSE of 0.00003, and an R^2 of 0.9429, showcasing its ability to model complex temporal patterns effectively.

Table 7: Percentage Improvement of KerasTuner-Optimized Stacked LSTM over ARIMA

Sentiment Class	RMSE (%)	MSE (%)	MAE (%)
Neutral	93.57	99.60	94.78
Positive	86.81	98.30	87.87
Negative	94.26	99.74	95.47

2. LSTM vs KerasTuner-Optimized Stacked LSTM Model Architecture

The LSTM model, which consists of a single hidden layer, is outperformed by the Stacked LSTM with KerasTuner across all sentiment classes, benefiting from the latter's deeper architecture and hyperparameter optimization.

For the **Neutral class** (Table 4), the LSTM model has an RMSE of 0.0266, MSE of 0.0007, MAE of 0.0204, and an R^2 of 0.8442. The Stacked LSTM with KerasTuner reduces the RMSE to 0.0079, MSE to 0.00006, and MAE to 0.0062, but its R^2 of 0.8667 is slightly higher than the LSTM's 0.8442.

For the **Positive class** (Table 5), the LSTM model reports an RMSE of 0.0536, MSE of 0.0029, MAE of 0.0436, and R^2 of 0.8534. The Stacked LSTM with KerasTuner improves these metrics, achieving an RMSE of 0.0132, MSE of 0.00017, MAE of 0.0102, and an R^2 of 0.9168, reflecting a better fit to the data.

The **Negative class** (Table 6) follows a similar pattern. The LSTM model has an RMSE of 0.0526, MSE of 0.0028, MAE of 0.0426, and R^2 of 0.8680, while the Stacked LSTM with KerasTuner achieves an RMSE of 0.0062, MSE of 0.00003, MAE of 0.0047, and an R^2 of 0.9429, demonstrating significant improvement in predictive accuracy.

3. Stacked LSTM vs KerasTuner-Optimized Stacked LSTM Model Architecture

The Stacked LSTM model, which consists of multiple LSTM layers, serves as a strong baseline. However, the Stacked LSTM with KerasTuner, which benefits from hyperparameter optimization, shows varied performance across the sentiment classes.

Table 8: Percentage Improvement of Stacked LSTM with KerasTuner over LSTM

Sentiment Class	RMSE (%)	MSE (%)	MAE (%)
Neutral	70.30	91.43	69.61
Positive	75.37	94.14	76.61
Negative	88.21	98.93	88.97

For the Neutral class (Table 4), the Stacked LSTM model achieves an RMSE of 0.0072, MSE of 0.0001, MAE of 0.0058, and an R^2 of 0.9167. The Stacked LSTM with KerasTuner, however, has a slightly higher RMSE of 0.0079, a lower MSE of 0.00006, a higher MAE of 0.0062, and a lower R^2 of 0.867. This indicates that while the SOTA model reduces the MSE, it does not consistently outperform the baseline in terms of RMSE, MAE, or R^2 for this class.

For the Positive class (Table 5), the Stacked LSTM model reports an RMSE of 0.0148, MSE of 0.00022, MAE of 0.0119, and R^2 of 0.9049. The Stacked LSTM with KerasTuner improves these metrics, achieving an RMSE of 0.0132, MSE of 0.00017, MAE of 0.0102, and an R^2 of 0.9168, reflecting a better overall performance.

The Negative class (Table 6) shows a similar trend. The Stacked LSTM model has an RMSE of 0.0085, MSE of 0.0001, MAE of 0.0069, and R^2 of 0.9113, while the Stacked LSTM with KerasTuner achieves an RMSE of 0.0062, MSE of 0.00003, MAE of 0.0047, and an R^2 of 0.9429, demonstrating significant improvement in predictive accuracy.

Table 9: Percentage Improvement of Stacked LSTM with KerasTuner over Stacked LSTM

Sentiment Class	RMSE (%)	MSE (%)	MAE (%)
Neutral	-9.72	40.00	-6.90
Positive	10.81	22.73	14.29
Negative	27.06	70.00	31.88

Experimental Setup

1. Fine-tuning RoBERTa Architecture

The experiments were conducted in five distinct stages, each building on the previous to analyze and forecast sentiment scores for customer reviews. Below are the details of each experiment, including the dataset, baselines, model architectures, training methods, and evaluation metrics.

Abstract RoBERTa (Robustly Optimized BERT Pretraining Approach) is a transformer-based model that improves upon BERT by removing the Next Sentence Prediction task,

using more data, and training longer with larger batches. Despite having the same architecture as BERT, RoBERTa achieves superior performance across NLP tasks, including sentiment analysis, due to its enhanced training strategy.

Rationale for Fine-Tuning Due to the absence of labeled sentiment data, we utilized a pretrained transformer model—RoBERTa—to assign sentiment scores to the dataset. However, these scores could not be treated as ground truth without validation. To assess their reliability, we manually labeled a sample of approximately 300 data points and compared them against the RoBERTa-generated labels.

The evaluation metrics below demonstrate that the pretrained model's performance was suboptimal, motivating the need for fine-tuning.

F1 Score	Precision	Recall	Accuracy
0.72	0.86	0.67	0.67

Table 10: Performance of RoBERTa before fine-tuned

Fine-Tuning the Model The model was fine-tuned using a manually labeled dataset of 300 data points. Of these, 80% were used for training and validation, while the remaining 20% (60 data points) were reserved for evaluation. Despite the limited size of the dataset, the fine-tuned model demonstrated notable performance improvements, as reflected in the evaluation metrics below.

F1 Score	Precision	Recall	Accuracy
0.75	0.81	0.82	0.82

Table 11: Performance of RoBERTa after fine-tuned

Performance Comparison: Pre-Fine-Tuning vs. Post-Fine-Tuning For this evaluation task, the F1 score is the most critical metric, as both false positives and false negatives are equally important. Despite using only 300 manually labeled data points, the model demonstrated the following performance changes after fine-tuning:

Metric	Before	After	% Change
F1 Score	0.72	0.75	**+4.17%**
Precision	0.86	0.81	**-5.81%** (drop)
Recall	0.67	0.82	**+22.39%**
Accuracy	0.67	0.82	**+22.39%**

Table 12: Evaluation Metrics Before and After Fine-Tuning

These results indicate that even limited manual labeling and fine-tuning can significantly enhance model performance. This process played a key role in building a more reliable forecasting model.

2. Setup of ARIMA

We first applied the ARIMA (AutoRegressive Integrated Moving Average) model as a traditional statistical baseline. The `statsmodels` library in Python was used to implement ARIMA.

The model parameters (p, d, q) were selected using the Akaike Information Criterion (AIC), resulting in an ARIMA(3,0,3) configuration: $p = 3$ (autoregressive order), $d = 0$ (differencing order), and $q = 3$ (moving average order) for negative sentiment.

The model parameters (p, d, q) were selected using the Akaike Information Criterion (AIC), resulting in an ARIMA(3,0,2) configuration: $p = 3$ (autoregressive order), $d = 0$ (differencing order), and $q = 2$ (moving average order) for positive sentiment.

The model parameters (p, d, q) were selected using the Akaike Information Criterion (AIC), resulting in an ARIMA(2,0,0) configuration: $p = 2$ (autoregressive order), $d = 0$ (differencing order), and $q = 0$ (moving average order) for neutral sentiment.

Transition: ARIMA → LSTM

Why the shift? The transition from ARIMA to LSTM was driven by the need to model the non-linear, complex, and long-term dependencies in sentiment data, which ARIMA could not adequately capture. LSTM's ability to learn these patterns through its deep learning architecture made it a more effective choice for our forecasting task, setting the stage for further improvements with Stacked LSTM and KerasTuner-optimized models.

3. Setup of LSTM

Next, we implemented a Long Short-Term Memory (LSTM) model (Table 1) using TensorFlow/Keras. The LSTM model consisted of a single hidden layer with 64 units, followed by two dense layers (32 units and 1 unit, respectively). The input data was reshaped into sequences of 30 time steps to capture temporal dependencies. The model was trained for 20 epochs using the Adam optimizer and mean squared error (MSE)

Transition: LSTM → Deep Stack LSTM

Why the shift?

- The basic LSTM model showed decent performance but had limitations in capturing complex temporal patterns in sentiment trends.
- We wanted to improve the model's ability to learn hierarchical and deeper patterns in the data.

Advantages of Deep Stack LSTM over basic LSTM:

- Layer Depth: Adding more LSTM layers allows learning more abstract and temporal features.
- Better Performance:
  - For example, R^2 scores increased for all classes (Positive: 0.8534 to 0.9049).
  - RMSE and MAE dropped significantly, meaning better prediction accuracy and less error.

4. Setup of Stacked LSTM

Next, we implemented a Long Short-Term Memory (LSTM) model (Table 2) using TensorFlow/Keras. The LSTM model consisted of a single hidden layer with 64 units, followed by two dense layers (32 units and 1 unit, respectively). The input data was reshaped into sequences of 30 time steps to capture temporal dependencies. The model was trained for 20 epochs using the Adam optimizer and mean squared error (MSE) Following the LSTM, we developed a Stacked LSTM model to leverage a deeper architecture. This model included three LSTM layers: the first two layers had 64 units each and returned sequences, while the third layer had 64 units. Dropout layers (with a rate of 0.2) were added after each LSTM layer to prevent overfitting. The architecture concluded with two dense layers (8 units and 1 unit, respectively). The model was trained similarly to the LSTM, using 20 epochs, the Adam optimizer, and MSE loss.

Transition: Deep Stack LSTM → Deep Stack LSTM with Keras Tuner

Why the shift?

- While the Deep Stack LSTM gave a performance boost, it had fixed architecture and hyperparameters.
- You wanted to automatically fine-tune the architecture (e.g., number of units, layers, learning rate) to achieve optimal performance.

Advantages of Keras Tuner-enhanced model:

- Automated Hyperparameter Optimization: It systematically searches the best combination, removing trial-and-error tuning.
- Highest Performance:
  - R2 for Positive increased from 0.9049 → 0.9168
  - RMSE and MAE reached their lowest, e.g., MAE for Positive: 0.0119 to 0.0102

5. Setup of Stacked LSTM with KerasTuner

Next, we implemented a Long Short-Term Memory (LSTM) model (Table 3) using TensorFlow/Keras. The LSTM model consisted of a single hidden layer with 64 units, followed by two dense layers (32 units and 1 unit, respectively). The input data was reshaped into sequences of 30 time steps to capture temporal dependencies. The model was trained for 20 epochs using the Adam optimizer and mean squared error (MSE) Finally, we optimized the Stacked LSTM model using the KerasTuner framework to fine-tune hyperparameters, creating our state-of-the-art (SOTA) model. The search space included the number of units in the LSTM layers (ranging from 32 to 128), learning rate (from 1×10^{-2} to 1×10^{-5}) also the stack from 1 to 5. The final architecture consisted of two LSTM layers (128 units and 30 units, respectively), with dropout layers (rate of 0.2), followed by two dense layers (12 units and 1 unit). The model was trained for 20 epochs with the optimized hyperparameters, using the Adam optimizer and MSE loss.

Formulation

Sentiment Scoring with RoBERTa

Sentiment scores for each review were assigned using a pretrained RoBERTa model (`cardiffnlp/twitter-roberta-base-sentiment`). The model outputs raw scores (logits) for three sentiment classes—Positive, Neutral, and Negative—which are converted to probabilities using the softmax function. For a given review, let z_i represent the raw score for sentiment class $i \in \{\text{Positive, Neutral, Negative}\}$. The probability for each class is calculated as:

$$P(i) = \text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j \in \{\text{Positive, Neutral, Negative}\}} e^{z_j}}$$

The probabilities sum to 1:

$$P(\text{Positive}) + P(\text{Neutral}) + P(\text{Negative}) = 1$$

These probabilities were used as sentiment scores for each review. The daily average sentiment scores for each class were then computed to form three univariate time series: $S_{\text{Positive}}(t)$, $S_{\text{Neutral}}(t)$, and $S_{\text{Negative}}(t)$, where t represents the time step (day).

Time Series Forecasting Models

Four models were used to forecast the sentiment time series: ARIMA, LSTM, Stacked LSTM, and Stacked LSTM with KerasTuner. Their formulations are described below.

ARIMA The ARIMA (AutoRegressive Integrated Moving Average) model was used as a baseline for forecasting sentiment time series in this project. The general equation for an ARIMA(p, d, q) process, as applied to a time series y_t, is given by:

$$y'_t = C + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where:

- y'_t is the differenced time series after applying d levels of differencing to make the series stationary, i.e., $y'_t = (1 - L)^d y_t$, where L is the lag operator ($Ly_t = y_{t-1}$),
- C is a constant term,
- $\phi_1, \ldots, \phi_p$ are the autoregressive coefficients for the p lagged terms,
- $\theta_1, \ldots, \theta_q$ are the moving average coefficients for the q lagged error terms,
- ϵ_t is the white noise error term at time t,
- $\epsilon_{t-1}, \ldots, \epsilon_{t-q}$ are the past error terms.

LSTM The Long Short-Term Memory (LSTM) model, a type of recurrent neural network, was used to capture nonlinear temporal dependencies. For a time step t, the LSTM updates are governed by the following equations:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(Forget Gate)}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(Input Gate)}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(Candidate Cell State)}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \text{(Cell State Update)}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(Output Gate)}$$

$$h_t = o_t \odot \tanh(C_t) \quad \text{(Hidden State)}$$

where:

- x_t is the input at time t (sentiment score),
- h_{t-1} is the previous hidden state,
- C_{t-1} is the previous cell state,
- W_f, W_i, W_C, W_o and b_f, b_i, b_C, b_o are the weights and biases,
- σ is the sigmoid activation function,
- $\odot$ denotes element-wise multiplication.

The LSTM model used a single hidden layer with 64 units, processing sequences of 30 time steps.

Stacked LSTM The Stacked LSTM extends the single-layer LSTM by stacking multiple LSTM layers. For a Stacked LSTM with L layers, the hidden state of the l-th layer at time t, $h_t^{(l)}$, is computed as:

$$h_t^{(l)} = \text{LSTM}(h_t^{(l-1)}, h_{t-1}^{(l)}, C_{t-1}^{(l)})$$

where $h_t^{(0)} = x_t$ (the input), and each layer applies the LSTM equations above. In this project, the Stacked LSTM had three layers, each with 64 units, with dropout (rate 0.2) applied after each layer to prevent overfitting.

Stacked LSTM with KerasTuner The Stacked LSTM with KerasTuner follows the same formulation as the Stacked LSTM but optimizes hyperparameters using KerasTuner. The final architecture included two LSTM layers (128 units and 36 units, respectively), with dropout (rate 0.2), followed by dense layers. The optimization process tuned the number of units, learning rate, and batch size, but the core LSTM equations remain as above.

Future Scope

- Multi-product Comparison: Extend the model to forecast sentiment trends across a broader range of products, potentially enabling cross-product comparisons and identifying market trends.
- Granular Sentiment Analysis: Incorporate aspect-based sentiment analysis to capture user opinions on specific features (e.g., safety, design, durability) within a single product review.
- Multi-modal Analysis: Integrate image or video review data along with text reviews to build a holistic sentiment forecasting framework.
- Real-Time Deployment: Deploy the forecasting model in a real-time dashboard for vendors and sellers to monitor sentiment shifts and optimize inventory or marketing decisions accordingly.

Limitation

- Limited Manual Labels: The fine-tuned model was trained on only 300 labeled samples, which may not fully capture the diversity of review sentiments across all products.
- Model Interpretability: Deep learning models like LSTM and stacked architectures are inherently black-box, making them less interpretable compared to traditional models like ARIMA.
- Static Forecasting: Forecasts are based on historical sentiment trends without incorporating external events (e.g., promotions, recalls) that may affect future sentiment.

References

Hugging Face, 2023. *cardiffnlp/twitter-roberta-base-sentiment*. Available at: https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment.

Hochreiter, S., and Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780.

Box, G.E.P., Jenkins, G.M., Reinsel, G.C., and Ljung, G.M., 2015. *Time Series Analysis: Forecasting and Control*. 5th ed. Wiley.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., and Invernizzi, L., 2019. *KerasTuner*. Available at: https://keras.io/keras$_tuner$/.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V., 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*.

Appendix

Dataset Description

The dataset used in this project consists of Amazon reviews from the Baby Products category, originally provided in `.jsonl` (JSON Lines) format. Each review includes fields such as `timestamp`, `verified_purchase`, `parent_asin`, and the review text. After preprocessing, the dataset was filtered to include only verified purchases, and reviews were grouped by `parent_asin`. The product with the second-highest number of reviews was selected, resulting in a subset saved as `TopProduct_2.csv`.

Abbreviations

- **ARIMA** = Autoregressive Integrated Moving Average
- **LSTM** = Long Short-Term Memory
- **S-LSTM** = Stacked Long Short-Term Memory
- **S-LSTM(KT)** = Stacked Long Short-Term Memory with KerasTuner

Time Series Forecasting Metrics

The evaluation metrics used for time series forecasting are defined as follows:

- **Root Mean Squared Error (RMSE)**:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

- **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

- **Mean Absolute Error (MAE)**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

- R^2:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\hat{y}_i$ is the predicted value, y_i is the actual value, $\bar{y}$ is the mean of the actual values, and n is the number of observations.