CryptoGPT: A Decoder-only Transformer Model

Yiwen Zhao
Carnegie Mellon University
Pittsburgh, PA 15213
yiwenz5@andrew.cmu.edu

Yanjun Chen
Carnegie Mellon University
Pittsburgh, PA 15213
yanjunch@andrew.cmu.edu

1 Introduction

In this project, we aimed to develop an advanced predictive model, *CryptoGPT*, for forecasting cryptocurrency prices using a custom-built decoder-only transformer architecture. The motivation stemmed from the need for accurate and efficient prediction methodologies that can capture both short-term and long-term trends in highly volatile markets. With Long Short-term network model as our baseline, our proposed method leverages decoder-only transformer, across different temporal resolutions and market conditions.

The LSTM was optimized using Mean Squared Error (MSE) with the Adam optimizer, while CryptoGPT utilized Root Mean Squared Error (RMSE) with the AdamW optimizer. Our results, validated through RMSE, MAPE, and Pearson Correlation Coefficient, highlighted CryptoGPT's strengths in daily data predictions and LSTM's advantages in hourly data predictions, underscoring the importance of selecting the right model for specific market conditions.

2 Dataset, Task

2.1 Overview of Cryptocurrency Market Data Features

The dataset utilized in this study comprises data points recorded at discrete-time intervals, which are either hourly or daily. This structure provides structured snapshots of market dynamics, allowing us to observe and analyze patterns over time. The prices within these intervals are recorded as continuous values. This means that within any given interval, the prices have the flexibility to fluctuate freely, capturing the inherent volatility of the cryptocurrency market.

To delve deeper into the specifics of the financial time series data, we focused on several key components that are critical for analyzing price movements. These include the 'High/Low Values' and the 'Open/Close Values' of each trading interval:

- **High/Low Values**: These metrics capture the maximum and minimum price points reached during each specified time interval. The high value indicates the peak price a cryptocurrency reached, which is pivotal in understanding market optimism, while the low value provides insight into the lowest price, reflecting the market's pessimism during the interval.
- **Open/Close Values**: These values record the price at which the trading period began (open) and ended (close). These figures are essential for charting and for techniques such as candlestick analysis, which can give investors insight into market sentiment and potential price movements. The difference between the open and close values within a single interval can indicate the direction of market movement and is often used to infer trends.

2.2 Bitcoin Daily Transaction Data

In our comparative analysis, we employ the same dataset used in Seabe et al.'s paper (11), sourced from Yahoo Finance. This dataset encompasses five years of daily transaction data for Bitcoin (BTC),

spanning from January 1, 2018, to January 1, 2023. Comprising exactly 1,825 data points, it provides a comprehensive overview of Bitcoin's market behavior over this period. Utilizing this dataset allows us to establish a solid baseline for our model's performance by directly comparing our results with those found in Seabe et al.'s study (11). The daily transaction dataset is partitioned into training (80%) and testing (20%) sets to align with the methodology used in Seabe et al.'s paper (11). By aligning our data source with established research, we aim to provide a stringent test of our proposed modeling techniques against a recognized benchmark in cryptocurrency analysis.

2.3 Bitcoin and Ether Hourly Transaction Data

In our project, we extend our analysis to assess the adaptability and robustness of our proposed models under varying conditions by integrating hourly transaction data for both Bitcoin (BTC) and Ethereum (ETH). This data, sourced from the Binance API, spans from April 15, 2022, to April 15, 2024, yielding a comprehensive set of 17,520 records. To ensure our models can manage the volatility and frequency inherent in hourly data, we follow industry standards by dividing the dataset into distinct segments—training, validation, and testing. Specifically, the dataset is structured into 240 hours of validation data and 120 hours of testing data, which facilitates rapid iterations and effective hyperparameter tuning. This segmentation strategy reduces overfitting, enhancing model robustness across various cryptocurrencies and time scales. By examining shorter intervals, we gain insights into how our models react to quick market changes, which is crucial for real-time trading and effective risk management.

2.4 Task

Our objective is to develop and assess sophisticated deep learning and generative AI models, including LSTM and the novel CryptoGPT model, to forecast Bitcoin and Ethereum prices at daily and hourly intervals. This task entails collecting and preprocessing cryptocurrency price data, training the models to maximize accuracy, and validating them to guarantee their generalizability. We plan to evaluate the performance of these models by using error metrics like RMSE (Root Mean Square Error) and MAPE (Mean Absolute Percentage Error), as well as correlation analysis, to ascertain their effectiveness across various timeframes.

2.5 Description of Models

We used LSTM as our baseline model due to its effectiveness in handling sequential data and its strong track record in financial time series forecasting. Our study focused on two models for cryptocurrency price prediction: a baseline Long Short-Term Memory (LSTM) model and a custom decoder-only Transformer model, CryptoGPT. The LSTM model, based on Seabe et al. (11), featured a three-layer architecture with 150 neurons per layer, utilizing a `MinMaxScaler` for data normalization and early stopping to prevent overfitting.

In contrast, CryptoGPT leveraged multi-head attention, a feed-forward network, and layer normalization across six layers to robustly analyze historical sequences. Both models focused on key features like High, Open, Low, and Close prices to capture temporal dynamics in volatile market conditions.

We proposed the decoder-only Transformer model for its robustness in sequential data handling, particularly for autoregressive prediction, leveraging historical sequences and extensive context windows. Additionally, decoder-only Transformers offer adaptability and scalability, allowing for extensive pre-training and fine-tuning on specialized financial data to enhance performance.

2.6 Evaluation Metrics

The evaluation metrics used were Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and the Pearson Correlation Coefficient. RMSE highlights larger prediction errors, crucial in financial modeling due to their impact on investment decisions. MAPE expresses prediction accuracy as a percentage, providing a clear sense of error magnitude relative to actual values, which aligns closely with financial risk. The Pearson Correlation Coefficient measures linear correlation, essential in cryptocurrency forecasting, as a high value indicates the model's reliability in capturing and predicting price trends.

3 Related Work

The exploration of cryptocurrency price prediction has increasingly leveraged advanced methodologies from machine learning, deep learning, and sentiment analysis to address the significant market volatility.

Lamon et al. (2017) employ traditional supervised learning algorithms to analyze news and social media data, labeling it based on future price changes rather than sentiment, which successfully predicts significant price fluctuations for Bitcoin and Ethereum over a 67-day test period (4). Similarly, Radityo et al. (2017) explore four Artificial Neural Network methods for predicting Bitcoin's next-day closing value, finding that the backpropagation neural network method is the most accurate and efficient (10).

Building on these foundations, Hashish et al. (2019) propose a hybrid model combining Hidden Markov Models with Long Short-Term Memory networks to enhance the prediction of cryptocurrency price movements, especially in the context of Blockchain-based Machine-to-Machine payment platforms for the IoT sector (1). Mohapatra et al. (2019) introduce KryptoOracle, a real-time, adaptive cryptocurrency price prediction platform that uses Twitter sentiments and a Spark-based architecture to manage large data volumes and adapt continuously to new market conditions (6). Kim et al. (2021) assess the performance of LSTM and GRU models on two datasets with differing statistical properties for predicting the prices of Bitcoin, Ethereum, and Litecoin; they find that GRU models perform better in downward trends for BTC and ETH, while LSTM excels in upward trends, with mixed results for LTC (2). Serafini et al. (2020) analyze the influence of network sentiments on Bitcoin prices, using ARIMAX and RNN models, and discover that ARIMAX outperforms RNN, showing that sentiment significantly influences market predictions (12).

Further, Tanwar et al. (2021) develop a real-time applicable deep-learning hybrid model combining GRU and LSTM to predict Litecoin and Zcash prices, achieving higher accuracy than existing methods and motivated by the unique challenges of the volatile cryptocurrency market (13). Politis et al. (2021) apply a deep learning methodology to forecast cryptocurrency prices, specifically Ether, achieving up to 84.2% accuracy in both short- and long-term predictions, addressing the high volatility and complex influencing factors of cryptocurrencies (9).

Lastly, Seabe et al. (2023) evaluate three Recurrent Neural Network types—LSTM, GRU, and Bi-LSTM—for predicting the exchange rates of major cryptocurrencies (Bitcoin, Ethereum, Litecoin), finding that Bi-LSTM outperforms the others in accuracy, suggesting its potential utility for investors and highlighting areas for future research (11).

Despite technological advancements, challenges remain in handling complex data interactions and incorporating broad influencing factors such as investor psychology and macroeconomic indicators. Pele et al. (2023) demonstrate that cryptocurrencies can be distinctly categorized as a separate asset class from traditional assets like stocks and bonds, primarily due to their unique tail factor, utilizing dimensionality reduction and various classification methods, including the Maximum Variance Components Split (8). Koutmos (2023) investigates how investor sentiments impact Bitcoin prices, emphasizing the psychological factors that influence market trends (3).

These studies collectively reflect a shift toward integrating more complex and nuanced modeling techniques in cryptocurrency prediction, emphasizing the ongoing need for models that effectively synthesize various influencing factors to enhance the accuracy and reliability of financial market predictions in a volatile economic sector.

4 Approach

4.1 Baseline Approach: Long Short-Term Memory Network

Model Architecture Our study utilizes a baseline LSTM model as described by Seabe et al.(11), featuring a three-layer architecture. Each of these layers contains 150 neurons 1. This configuration was selected to replicate the baseline model.

Data Preparation In the preparation of our dataset, we ensured that there were no missing values, adhering to standard data quality practices. For normalizing the price data, we utilized a `MinMaxScaler`, applying it only to the training data to prevent data leakage from the test set. This preprocessing

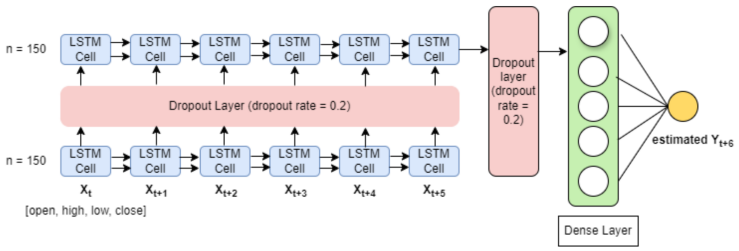


Figure 1: LSTM Architecture Illustration

approach is consistent with the methods outlined in the referenced study by Seabe et al., ensuring reliability in our data input.

Implementation Details The implementation of our LSTM model was performed using the TensorFlow Keras API. To combat the risk of overfitting, we integrated an early stopping mechanism during the training process. The model was trained using a batch size of 120, in alignment with the parameters employed in the original study by Seabe et al (11). Our model employs a look-back period of 5, determined through tuning, using data from the previous five time steps for predictions. This setup is crucial for capturing the temporal dynamics in cryptocurrency price movements. Due to the unavailability of source code and detailed network specifics from the original study, we made experimental adjustments to approximate the reported RMSE and MAPE metrics, enhancing our understanding of model performance under various configurations.

4.2 Main Method: Decoder-only Transformer Model

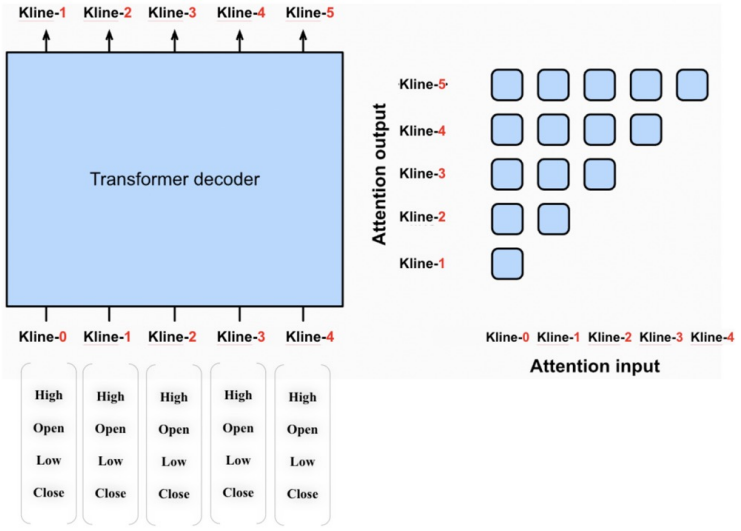


Figure 2: CryptoGPT (Image Source: Dive into Deep Learning(17))

Cryptocurrency Data Encoding

- **Data Features:** Cryptocurrency data include four crucial features: High, Open, Low, and Close prices.
- **Embedding Details:** This data is embedded into input sequences to enrich the model's price information, using an embedding size of 192.
- **Sequence Handling:** The model processes sequences with a length of 128, optimizing the amount of data each input batch carries.

Positional Encodings To maintain the integrity of data sequence order, positional encodings are embedded into the input streams. This preserves the chronological order, essential for the model to accurately interpret the temporal sequence of events.

Model Architecture Specifications We developed CryptoGPT, a decoder-only transformer model, from scratch, following the detailed architecture shown in Figure 2.

- **Multi-Head Attention:** The architecture employs six attention heads, allowing it to concurrently process multiple segments of the input for better pattern recognition and insights.
- **Feed-Forward Network:** After attention processing, a feed-forward neural network synthesizes the data to form predictions, transforming attention-driven insights into actionable outputs.
- **Layer Normalization:** Each transformer block includes layer normalization to ensure stable and efficient model training, crucial for maintaining performance consistency across training epochs.
- **Comprehensive Layer Design:** The model is constructed with six layers, each integrating attention, feed-forward networking, and normalization components to robustly learn from complex data sequences.

5 Experiments

5.1 Experimental Design Overview

Our experimental design aimed to evaluate and compare the predictive capabilities of architectures, an LSTM model and a custom-built decoder-only Transformer model named CryptoGPT, in forecasting cryptocurrency prices.

The study focused on several datasets including Bitcoin daily and hourly price data, and Ethereum hourly price data, to assess the models' ability to capture temporal dynamics, generalize across different cryptocurrencies, and evaluate the impact of data granularity on prediction accuracy.

Both models were trained using distinct error optimization metrics: the LSTM employed Mean Squared Error (MSE) with the Adam optimizer, a learning rate of 0.01, enhanced by a learning rate scheduler and early stopping, and a batch size of 120. CryptoGPT used Root Mean Squared Error (RMSE) with the AdamW optimizer at a learning rate of 0.005 and a batch size of 64.

Key performance metrics such as RMSE, MAPE, and Pearson Correlation Coefficient quantified prediction accuracy and model reliability.

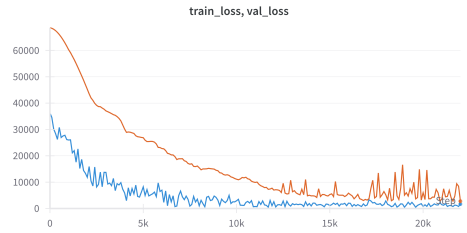
5.2 Timing Information

The experiments involved tailored hardware configurations to meet the distinct computational needs of each model. The decoder-only Transformer model was equipped with NVIDIA A100 GPUs and took advantage of the high RAM capacity provided by Google Colab, reflecting its need for substantial computational power. In contrast, the LSTM model was trained using standard CPUs and system memory, aligning with its lower computational demands. This setup ensured that both models operated optimally within their respective requirements for processing power and memory.

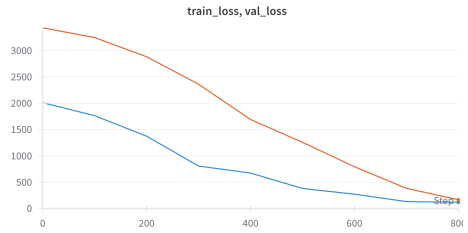
Training Time:

- **Epoch Duration:** The LSTM models for the BTC and ETH Hourly Price Datasets were trained for 50 epochs, each lasting about 8 seconds. For the BTC Daily Price Dataset, training duration was shorter at approximately 1.1 seconds per epoch due to its smaller data size. CryptoGPT does not utilize epochs but rather iterations to manage training.
- **Convergence Time:** For LSTM, early stopping was triggered at the 31st and 27th epochs for the BTC and ETH Hourly datasets respectively, resulting in a total training time of approximately 5 minutes. The BTC Daily dataset achieved convergence in just 40 seconds. CryptoGPT's training varied, taking about 2 minutes for the ETH Hourly dataset, 40 minutes for the BTC Hourly dataset, and 10 minutes for the BTC Daily dataset to reach convergence.

We also provide the evolution of training loss over time for both the Long Short-Term Memory (LSTM) model and the custom-built decoder-only Transformer model, CryptoGPT. The figures 3 and 4 illustrate how each model's training loss decreases as training progresses, reflecting the learning efficiency and convergence behavior of the models.

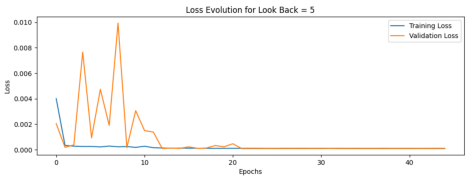


(a) Training Loss of CryptoGPT Model on BTC Hourly Data

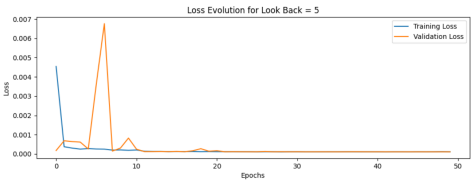


(b) Training Loss of CryptoGPT Model on ETH Hourly Data

Figure 3: Training Loss of CryptoGPT Model on Hourly Data Over Time



(a) Training Loss of LSTM Model on BTC Hourly Data



(b) Training Loss of LSTM Model on ETH Hourly Data

Figure 4: Training Loss of LSTM Model on Hourly Data Over Time

5.3 Experimental Results

Table 1: Comparison of RMSE and MAPE (%) Metrics for Baseline Replicated Model, and CyptoGPT on Daily Bitcoin Transaction Data

	RMSE	**MAPE(%)**
LSTM Replication	1452.39	3.46
LSTM Baseline (Seabe et al., 2023)	1031.34	3.94
CyptoGPT Model	600.39	2.28

Daily Data Performance (Table 1): Our replicated LSTM model closely aligns with the metrics achieved in the original paper, serving as a benchmark for further experiments. CryptoGPT Model significantly outperforms the LSTM models with the lowest RMSE (600.39) and MAPE (2.28%), indicating superior accuracy in capturing daily Bitcoin transaction patterns. LSTM Models is Less effective with higher RMSE and MAPE compared to CryptoGPT, as noted in both replication and Seabe et al. (2023) models.

Hourly Data Performance (Table 3; Table 2; Figure 5): LSTM Models excel in handling hourly data for both Bitcoin and Ethereum, showing lower RMSE and MAPE, suggesting better handling of frequent fluctuations. CryptoGPT Model shows decreased performance with higher RMSE and MAPE, struggling with the high volatility of hourly data.

Correlation Analysis (Table 4): LSTM Model maintains high correlation coefficients (above 0.94), indicating strong alignment with actual price movements. CryptoGPT Model exhibits lower correlations, especially notable in hourly Bitcoin data.

Table 2: Comparison of RMSE Values for Baseline and CryptoGPT on Hourly Cryptocurrency Data

Model / Cryptocurrency	Bitcoin (Hourly)	Ethereum (Hourly)
LSTM Baseline (Replication)	812.28	53.82
CryptoGPT Model	3329.99	97.228

Table 3: Comparison of MAPE(%) Values for Baseline and CryptoGPT on Hourly Cryptocurrency Data

Model / Cryptocurrency	Bitcoin (Hourly)	Ethereum (Hourly)
LSTM Baseline (Replication)	0.72	1.01
CryptoGPT Model	3.29	2.58

5.4 Key Insights

CryptoGPT's Strengths and Limitations: CryptoGPT is particularly effective for predicting daily cryptocurrency data, where its capacity to analyze long-term trends shines due to a global attention mechanism that comprehensively evaluates the entire input sequence. This ability enables the model to extract and leverage broad contextual information, which is crucial for understanding overarching market movements. However, the same global attention mechanism that benefits daily data analysis can become a limitation when applied to hourly data. In these cases, CryptoGPT may overfit on sporadic trends and become overly sensitive to noise, diminishing its effectiveness in more granular time frames where minute-by-minute changes are less about broad trends and more about immediate market responses.

LSTM's Advantages: LSTMs excel in scenarios requiring acute awareness of short-term dependencies, thanks to their sequential data processing capabilities which are adept at filtering out noise and focusing on relevant temporal sequences. This makes them particularly suited for real-time or near-real-time applications, like hourly cryptocurrency price predictions, where market conditions can change swiftly and demand a rapid analytical response based on the most recent data. Their architecture enables them to focus effectively on the immediate past, making them invaluable tools for tracking and responding to fast-paced market dynamics.

6 Code Overview

6.1 Development of LSTM Model: Replicating the Selected Baseline

Since the LSTM model follows the same procedure for daily Bitcoin, hourly Ether, and hourly Bitcoin predictions, we will present a unified code overview for the hourly Bitcoin model to minimize redundancy. This section offers a detailed overview of the code used to develop and evaluate the LSTM model, based on our selected baseline.

1. **Time Tracking Function (Figure 6):** This function is used to monitor the time spent on various steps of model training, ensuring efficient time management throughout the development process.
2. **Data Reading (Figure 7):** Displays the code used to read cryptocurrency data, specifically hourly Bitcoin prices, which is the first step in our data pipeline.
3. **Data Preprocessing (Figure 8):** This snippet shows how the cryptocurrency data is preprocessed, including normalization and missing value check, to make it suitable for training the LSTM model.
4. **Dataset Creation (Figure 9):** Illustrates the function that constructs the dataset for LSTM training, structuring the data into sequences that the model can learn from.
5. **MAPE Calculation Function (Figure 10):** This function calculates the Mean Absolute Percentage Error (MAPE), a critical metric for evaluating the accuracy of our predictions.
6. **LSTM Model Construction (Figure 11):** Shows the function used to create the LSTM model architecture, detailing the layers and configurations employed.

Table 4: Comparison of Pearson Correlation Coefficients for Baseline and CryptoGPT Models on Bitcoin and Ethereum

Model / Cryptocurrency	Bitcoin (Daily)	Bitcoin (Hourly)	Ethereum (Hourly)
LSTM Baseline (Replication)	0.9896	0.9418	0.9643
CryptoGPT Model	0.9596	0.7592	0.9209

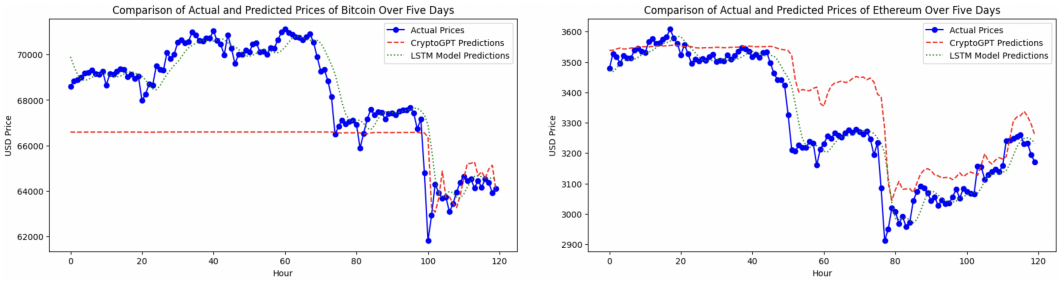


Figure 5: Experiment Result on Test Set

7. **Callbacks Creation (Figure 12):** Depicts the function used to create callbacks for the LSTM model training. Callbacks such as early stopping and model checkpointing are crucial for enhancing the training process by preventing overfitting and ensuring that the best model state is saved.
8. **Model Training Function (Figure 13):** Displayed here is two functions responsible for training and evaluating the LSTM model respectively.
9. **Training Process (Figure 14):** Provides insight into the actual training process, capturing live metrics and adjustments during each epoch.
10. **RMSE Display (Figure 15):** Shows the function used to display the Root Mean Square Error (RMSE) results, another vital metric for assessing model performance.
11. **Pearson Correlation Coefficient Calculation (Figure 16):** This snippet calculates the Pearson Correlation Coefficient, helping us understand the linear relationship between predicted and actual prices.
12. **Results Saving (Figure 17):** Finally, this code saves the results in a pickle file for later comparison and analysis, ensuring that all data is securely stored for future reference.

6.2 CryptoGPT

This section outlines the structure and functionality of various Python scripts used in the development of a GPT model for cryptocurrency analysis, utilizing PyTorch and WandB:

1. **transformer/cryptogpt.py**: this notebook provides a comprehensive example of training a GPT model for cryptocurrency analysis, demonstrating data preprocessing, model configuration, training, and evaluation using PyTorch and WandB for monitoring and analysis.

   From Line 60 - 104, we define a custom dataset class (CryptoDataset) for handling cryptocurrency data and implement methods to retrieve data samples and their corresponding labels, considering features like open, high, low, close prices, and volume.

   From Line 171 - 259, we implement batch end callback and epoch end callback. This method utilizes callbacks to perform specific actions at different stages of training, such as batch end and epoch end and enables logging of training losses, attention computation times, and memory consumption.
2. **transformer/cryptogpt/trainer.py**: This file defines a generic training loop that can be applied to any neural network, providing functionalities for training, validation, and testing. From Line 92 - 114, we implement train loader, val loader, and test loader.
3. **transformer/cryptogpt/model.py**: The provided file implements a GPT (Generative Pre-trained Transformer) language model. It integrates various components necessary for training, evaluating, and generating sequences using a GPT language model, with options

for customizing attention mechanisms and positional encoding techniques. From Line 472 - 523, we evaluate the trained model on validation and test datasets using provided evaluation metrics such as R-squared (R2) and mean absolute percentage error (MAPE).

7 Timeline

The table 5 summarizes the distribution of time spent on various activities during the project:

Activity	Time Spent (hours)
Reading papers and resources	10
Exploring datasets and sources	4
Reviewing documentation (e.g., PyTorch)	3
Understanding existing implementations	2
Replicating baseline model code	15
Writing new code from scratch	30
Writing scripts for experiments	20
Running experiments	5
Compiling and analyzing results	20
Writing the final document	15
Total	**124**

Table 5: Summary of Time Spent on Project Activities

8 Research Log

Our project began with an ambitious goal to advance cryptocurrency price prediction using advanced AI techniques, specifically through decoder-only transformer models. Initially, we aimed to harness these models to enhance accuracy and efficiency across various transaction intervals. As we progressed, our journey took several turns, mainly driven by the challenges associated with the complex and volatile nature of cryptocurrency data, which led to issues with model stability and extended training times.

The key challenge was managing the transformer model's sensitivity to the inherent noise and rapid changes in cryptocurrency data. This required us to deviate from our initial plan, prompting extensive experimentation with different architectural configurations. We adjusted the layer setups and attention mechanisms to better handle the unpredictable data patterns, striving to stabilize the training process.

This iterative process of adapting our approach was crucial, not only in addressing the gaps in our initial methodology but also in refining our overall strategy. Each adaptation brought us closer to a model that could reliably handle the complexities of financial time series data. By continuously tweaking and testing our models, we developed a more robust solution that could effectively predict cryptocurrency prices, showcasing our team's resilience and innovative capacity in overcoming the unpredictabilities of AI-driven financial analysis.

9 Conclusion

In our study, we aimed to enhance cryptocurrency price prediction using advanced AI models, focusing on the baseline LSTM and the novel CryptoGPT. We found that CryptoGPT excelled in daily predictions, leveraging its global attention mechanism for effective long-term trend analysis, while LSTM showed resilience in handling hourly data, ideal for rapid market fluctuations.

Future work should include increasing data volume from 2 to 5 or 10 years for improved robustness, integrating sentiment from social media to capture market sentiment, exploring inter-correlations among cryptocurrencies using the Attention mechanism, and developing a hybrid LSTM and Transformer model to leverage both short-term dependencies and broad context processing for enhanced predictions.

References

[1] I. A. Hashish, F. Forni, G. Andreotti, T. Facchinetti, & S. Darjani. A Hybrid Model for Bitcoin Prices Prediction using Hidden Markov Models and Optimized LSTM Networks. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain, pp. 721-728, 2019, doi: 10.1109/ETFA.2019.8869094.

[2] J. Kim, S. Kim, H. Wimmer, & H. Liu. A cryptocurrency prediction model using LSTM and GRU algorithms. In *Proc. IEEE/ACIS 6th Int. Conf. Big Data, Cloud Comput., Data Sci. (BCD)*, pp. 37-44, Sep. 2021.

[3] Dimitrios Koutmos. Investor sentiment and bitcoin prices. *Review of Quantitative Finance and Accounting*, 60, 1-29, 2023. https://doi.org/10.1007/s11156-022-01086-4.

[4] C. Lamon, E. Nielsen, & E. Redondo. Cryptocurrency price prediction using news and social media sentiment. *SMU Data Science Review*, vol. 1, no. 3, pp. 1-22, 2017.

[5] Y. Ma, C. Ventre, and M. Polukarov. Denoised Labels for Financial Time-Series Data via Self-Supervised Learning. arXiv preprint arXiv:2112.10139, 2021.

[6] S. Mohapatra, N. Ahmed, & P. Alencar. KryptoOracle: A Real-Time Cryptocurrency Price Prediction Platform Using Twitter Sentiments. In *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, pp. 5544-5551, 2019. doi: 10.1109/BigData47090.2019.9006554.

[7] B. Aditya Pai, L. Devareddy, S. Hegde, and B. S. Ramya. A Time Series Cryptocurrency Price Prediction Using LSTM. In *Emerging Research in Computing, Information, Communication and Applications*, pages 653–662, 2021. `https://doi.org/10.1007/978-981-16-1342-5_50`.

[8] Daniel Traian Pele, Niels Wesselhöfft, Wolfgang Karl Härdle, Michalis Kolossiatis, & Yannis G. Yatracos. Are cryptos becoming alternative assets? *The European Journal of Finance*, 29:10, 1064-1105, 2023, DOI: 10.1080/1351847X.2021.1960403.

[9] A. Politis, K. Doka, & N. Koziris. Ether price prediction using advanced deep learning models. In *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, pp. 1-3, May 2021.

[10] A. Radityo, Q. Munajat, & I. Budi. Prediction of Bitcoin exchange rate to American dollar using artificial neural network methods. In *Proc. Int. Conf. Adv. Comput. Sci. Inf. Syst. (ICACSIS)*, pp. 433-438, 2017, doi: 10.1109/ICACSIS.2017.8355070.

[11] P. L. Seabe, C. R. B. Moutsinga, and E. Pindza. Forecasting cryptocurrency prices using LSTM, GRU, and bi-directional LSTM: A deep learning approach. *Fractal and Fractional*, 7(2):203, 2023.

[12] G. Serafini, P. Yi, Q. Zhang, M. Brambilla, J. Wang, Y. Hu, & B. Li. Sentiment-Driven Price Prediction of Bitcoin Based on Statistical and Deep Learning Approaches. In *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, pp. 1-8, 2020, http://dx.doi.org/10.1109/IJCNN48605.2020.9206704.

[13] S. Tanwar, N. P. Patel, S. N. Patel, J. R. Patel, G. Sharma, & I. E. Davidson. Deep learning-based cryptocurrency price prediction scheme with inter-dependent relations. *IEEE Access*, 9, 138633-138646, 2021.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems*, 30, 2017.

[15] Z. Zeng, R. Kaur, S. Siddagangappa, S. Rahimi, T. Balch, and M. Veloso. Financial Time Series Forecasting using CNN and Transformer. arXiv preprint arXiv:2304.04912, 2023.

[16] H. Zhao, et al. Revolutionizing Finance with LLMs: An Overview of Applications and Insights. arXiv preprint arXiv:2401.11641, 2024.

[17] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

A Appendix

A.1 Code Snapshots

A.1.1 Development of LSTM Model: Replicating the Selected Baseline

Functions to track time spent

```
[ ] from keras.callbacks import Callback

    class TimeHistory(Callback):
            def on_train_begin(self, logs={}):
                    self.times = []

            def on_epoch_begin(self, epoch, logs={}):
                    self.epoch_start_time = time.time()

            def on_epoch_end(self, epoch, logs={}):
                    self.times.append(time.time() - self.epoch_start_time)
```

Figure 6: Funtion to track time spent

Read Data from Google Drive

```
file_path = '/content/drive/MyDrive/CMU10623/Team Project/BTCUSDT_kline_1h_22_10-24_14'

# Read the CSV file directly using its full path
df = pd.read_csv(file_path)
df = df.reset_index(drop=True)
df.drop('Unnamed: 0', axis=1, inplace=True)
# Display the first few rows of the DataFrame to confirm it's loaded correctly
print(df.head())
print(df.columns)
```

Figure 7: Code to read cryptocurrency (bitcoin hourly) data

Data Preprocessing

```
[ ] # Ensure DataFrame is sorted by time
    df['open_time'] = pd.to_datetime(df['open_time'])
    df.sort_values('open_time', inplace=True)
```

```
[ ] missing_values_count = df.isna().sum()
    print(missing_values_count)
```

```
open_time                    0
open                         0
high                         0
low                          0
close                        0
volume                       0
close_time                   0
quote                        0
number_of_trades             0
Taker buy base asset volume  0
Taker buy quote asset volume 0
Unused field                 0
dtype: int64
```

```
[ ] # Define your train, validation, and test sizes
    val_size = 24 * 5 * 2
    test_size = 24 * 5
    train_size = int(len(df)) - val_size - test_size
```

Figure 8: Code to pre-process crypto data

Create dataset for LSTM network training

```
[ ] def create_dataset(dataset, look_back):
        X, Y = [], []
        for i in range(len(dataset) - look_back):
            a = dataset[i:(i + look_back), 0:4]    # Features: Open, High, Low, Close
            X.append(a)
            Y.append(dataset[i + look_back, 3])    # Target: Next hour Close
        return np.array(X), np.array(Y)
```

Figure 9: Function to create dataset for LSTM training

Evaluation metrics

```
[ ] def calculate_mape(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        non_zero = y_true != 0
        return np.mean(np.abs((y_true[non_zero] - y_pred[non_zero]) / y_true[non_zero])) * 100
```

Figure 10: Function to create MAPE metrics

Create model for LSTM network

```
def create_model_dropout(input_shape, units=100, dropout_rate=0.2):
    model = Sequential([
        LSTM(units, activation='tanh', recurrent_activation='sigmoid', return_sequences=True, input_shape=input_shape),
        Dropout(dropout_rate),
        LSTM(units, activation='tanh', recurrent_activation='sigmoid'),
        Dropout(dropout_rate),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')
    return model
```

Figure 11: Function to create LSTM model

Set up callbacks during model training

```
[ ] # Set up the early stopping callback
    early_stopping = EarlyStopping(
        monitor='val_loss',    # Monitor validation loss
        patience=10,           # Number of epochs with no improvement after which training will be stopped
        verbose=1,             # To log the epoch at which training stops
        mode='min',            # Stops training when the quantity monitored has stopped decreasing
        restore_best_weights=True    # Restores model weights from the epoch with the best value of the monitored quantity
    )
```

```
# Set up the Reduce Learning Rate callback
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.1,            # Factor by which the learning rate will be reduced. new_lr = lr * factor
    patience=5,            # Number of epochs with no improvement after which learning rate will be reduced
    verbose=1,             # Int to print out log messages when reducing LR
    mode='min',            # The quantity to be monitored must decrease for the LR to reduce
    min_delta=0.0001,      # Threshold for measuring new optimums
    cooldown=0,            # Number of epochs to wait before resuming normal operation after LR has been reduced
    min_lr=0               # Lower bound on the learning rate
)
```

Figure 12: Function to create callbacks for LSTM training

Model Training and Evaluation

```
[ ] def train_model(model, X_train, y_train, X_test, y_test, epochs=50, batch_size=120):
        time_callback = TimeHistory()
        early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
        reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10)

        history = model.fit(
            X_train, y_train, epochs=epochs, batch_size=batch_size,
            validation_data=(X_test, y_test),
            callbacks=[early_stopping, reduce_lr, time_callback],
            verbose=1
        )
        return model, history, time_callback.times


    def evaluate_model(y_true, y_pred):
        rmse = np.sqrt(mean_squared_error(y_true, y_pred))
        mape = calculate_mape(y_true, y_pred)
        return rmse, mape
```

Figure 13: Function to train and evaluate the LSTM model

```
look_back_values = [5, 10, 20, 30]
results = {}
for look_back in look_back_values:
    # Split the DataFrame into training, validation, and test sets
    train_df = df[:train_size]
    val_df = df[(train_size - look_back):(train_size + val_size)]
    test_df = df[(train_size + val_size - look_back):]

    # Fit the scaler on the training data only
    scaler = MinMaxScaler()
    scaler.fit(train_df[['open', 'high', 'low', 'close']])

    # Transform both training and test data with the fitted scaler
    df_train_scaled = scaler.transform(train_df[['open', 'high', 'low', 'close']])
    df_val_scaled = scaler.transform(val_df[['open', 'high', 'low', 'close']])
    df_test_scaled = scaler.transform(test_df[['open', 'high', 'low', 'close']])

    X_train, y_train = create_dataset(df_train_scaled, look_back)
    X_val, y_val = create_dataset(df_val_scaled, look_back)
    X_test, y_test = create_dataset(df_test_scaled, look_back)
    print(len(X_val))
    print(len(X_test))

    # Define the model
    model = create_model_dropout(input_shape=(X_train.shape[1], X_train.shape[2]), units=150, dropout_rate=0.2)

    # Time the training process
    start_time = time.time()
    model, history, times_per_epoch = train_model(model, X_train, y_train, X_val, y_val)
    training_time = time.time() - start_time  # Total time for training until convergence
    average_epoch_time = np.mean(times_per_epoch)


    # Predict using the model
    start_time = time.time()
    predicted_prices = model.predict(X_test)
    validation_time = time.time() - start_time  # Time for validation

    predicted_prices = scaler.inverse_transform(np.column_stack((np.zeros((len(predicted_prices), 3)), predicted_prices)))[:, 3]
    actual_prices = scaler.inverse_transform(np.column_stack((np.zeros((len(y_test), 3)), y_test)))[:, 3]

    # Evaluate the model
    rmse, mape = evaluate_model(actual_prices, predicted_prices)

    # Store results in the dictionary
    results[look_back] = {
        'predicted': predicted_prices,
        'actual': actual_prices,
        'history': history,
        'rmse': rmse,
        'mape': mape,
        'training_time': training_time,
        'average_epoch_time': average_epoch_time,
        'validation_time': validation_time
    }
```

Figure 14: LSTM model training process

```
# Create plots for each look_back value
fig, axs = plt.subplots(len(look_back_values), 1, figsize=(10, 8), sharex=True)

for i, look_back in enumerate(look_back_values):
    axs[i].plot(results[look_back]['actual'], label='Actual Prices')
    axs[i].plot(results[look_back]['predicted'], label='Predicted Prices', linestyle='--')
    axs[i].set_title(f'Look Back = {look_back}, RMSE = {results[look_back]["rmse"]:.2f}, MAPE = {results[look_back]["mape"]:.2f}%')
    axs[i].legend()

# Add common labels
fig.suptitle('Comparison of Actual and Predicted Prices for Different Look Back Values')
plt.xlabel('Time Steps')
plt.ylabel('Price')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

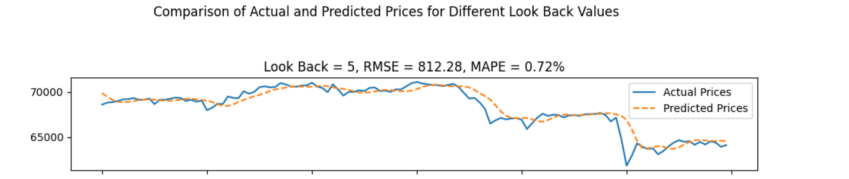


Figure 15: Check RMSE values

```
for i, look_back in enumerate(look_back_values):
    correlation_coefficient, p_value = pearsonr(results[look_back]['predicted'], results[look_back]['actual'])
    print("Look back: ", look_back)
    print("Pearson Correlation Coefficient:", correlation_coefficient)
    print("P-value:", p_value)
```

```
Look back:  5
Pearson Correlation Coefficient: 0.9418642063093319
P-value: 9.947560344367731e-58
Look back:  10
Pearson Correlation Coefficient: 0.9046467056539026
P-value: 1.5763701043730657e-45
Look back:  20
Pearson Correlation Coefficient: 0.9141841959294947
P-value: 4.176820949735782e-48
Look back:  30
Pearson Correlation Coefficient: 0.8852413149619839
P-value: 4.909326368092706e-41
```

Figure 16: Calculate Pearson Correlation Coefficient values

```
with open('/content/drive/My Drive/CMU10623/Team Project/lstm_btc_hour_train_val_test_results.pkl', 'wb') as handle:
    pickle.dump(results, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Figure 17: Save results in pickle file for later comparison analysis