Predicting Bitcoin Returns

Project by: Kimble Horsak, Ashley Hattendorf, Janie Chen, Sarah Lee, and Gayathree Gopi
Project Links: Bitcoin-Prediction-Code

Abstract

Cryptocurrency markets, led by Bitcoin, are known for their high volatility and unpredictability, presenting significant challenges for price prediction. This project explores the feasibility of forecasting Bitcoin's daily percentage returns using advanced machine learning techniques. By integrating historical price data with financial indicators such as SMA, EMA, and squeeze metrics, as well as sentiment data from GDELT, we aim to create a model that captures the intricate dynamics of cryptocurrency markets. The project utilizes Bi-Directional Long Short-Term Memory (LSTM) networks, which are well-suited for time series prediction, and employs advanced hyperparameter tuning techniques like HyperOpt to optimize model performance. Key evaluation metrics, including Mean Squared Error (MSE) and R^2, are used to assess prediction accuracy and robustness.

Our contributions include a custom loss function, "NoSmallLoss," designed to address trivial predictions, and the implementation of dynamic pruning to enhance computational efficiency during hyperparameter tuning. Despite the challenges, our findings highlight the importance of predicting returns instead of prices to better reflect market behavior. While sentiment analysis provided limited predictive power, incorporating extreme sentiment days showed potential for anomaly-driven trading strategies. This project serves as a stepping stone for future research in applying machine learning to financial markets, emphasizing the need for scalable methods and innovative approaches to handle the complexities of cryptocurrency price prediction.

Introduction

Cryptocurrency markets, led by Bitcoin, are rapidly evolving and drawing attention due to their high volatility and potential for massive returns. However, predicting cryptocurrency returns is a highly challenging task due to the market's volatility and susceptibility to global macroeconomic trends. Designing an algorithm capable of navigating these complexities could help retail investors make informed decisions, mitigate risks, and even stabilize the market by reducing reliance on emotion-driven trading.

Despite the fact that it's nearly impossible for traders to create a reliable predictive model using advanced neural networks, we aim to explore this challenge. The goal is not only to test the feasibility of such predictions but also to deepen our understanding of time series modeling and the complexities of cryptocurrency markets.

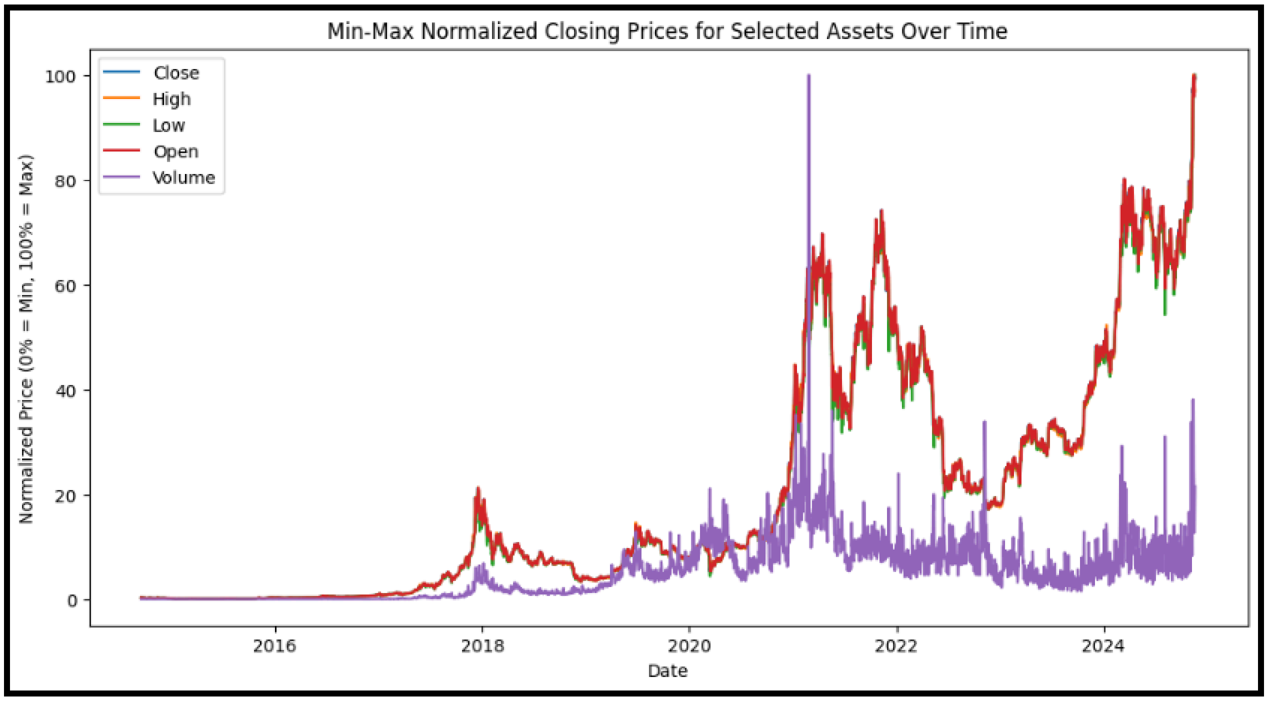


Figure 1: Graphical representation of Bitcoin

As we can see, Bitcoin has truly taken off in the last 10 years making this project more relevant than ever.

Approach and Rationale

This project aims to tackle the challenging task of predicting changes in Bitcoin prices by combining historical price data with sentiment analysis. By leveraging data from Yahoo Finance (yfinance), we plan to utilize key financial indicators like Squeeze metrics and both exponential and simple moving averages. These indicators will provide a well-rounded view of market activity and trends. Rather than predicting absolute prices, which can be misleading and a common pitfall in these types of predictions, we'll focus on forecasting daily percentage returns.

We plan to utilize time series prediction models, such as Long Short-Term Memory (LSTM) networks to tackle this challenge. LSTMs are suited for this task because they excel at capturing patterns and relationships over time. To enhance the model's accuracy and robustness, we'll integrate additional data sources, such as social media sentiment, which reflects market psychology and real-time reactions to events.

The model's performance will be evaluated using metrics such as Mean Squared Error (MSE) and R^2 to measure prediction accuracy to assess how well the model predicts returns. By blending traditional financial indicators with modern machine learning and sentiment analysis, this project seeks to understand and predict Bitcoin's volatile price dynamics.

Our Unique Perspective

Our project includes a custom loss function, “NoSmallLoss,” to address the issue of models predicting zero values, and employs dynamic pruning techniques to optimize resources during hyperparameter tuning. Additionally, the use of sentiment data and the application of advanced dropout techniques enhance model robustness. This project serves as both a learning opportunity for time series analysis and a potential tool for traders to navigate the volatile cryptocurrency market.

Data Collection

Before going into the specifics of our models and all the complicated transformations we made, it is important to define key variables used in our analyses.

Squeeze

Squeeze is a popular binary volatility and momentum indicator used by traders in various financial markets. This indicator is designed to help traders identify potential “squeeze” scenarios where an asset is likely to experience a significant increase in volatility, often leading to a strong directional move, and momentum might soon increase significantly.

For example:
“On” (1): Market is likely to gain momentum (trend is forming).
“Off” (0): Market has low momentum (no clear trend).

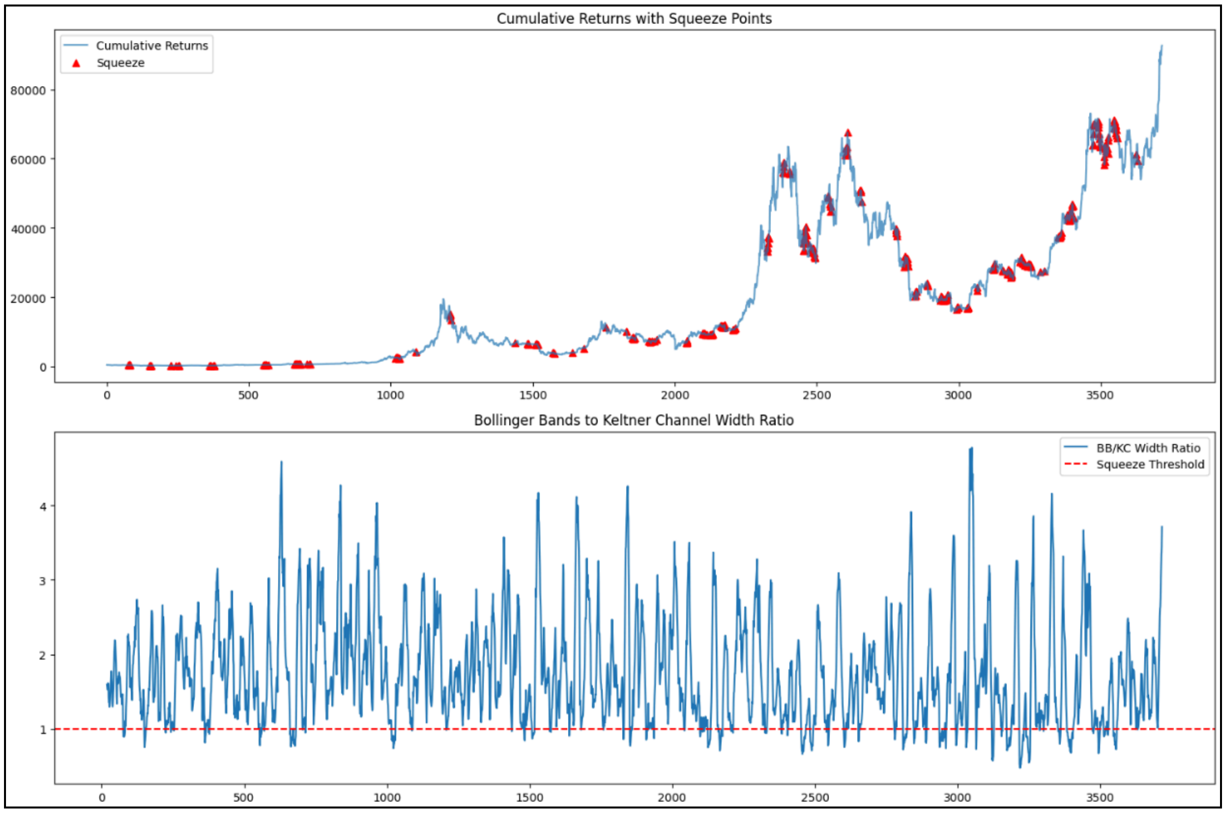


Figure 2: Graphical Depiction of Squeeze

The top chart depicts Cumulative Returns over time (blue line) with squeeze points highlighted by red triangles. Notice how red triangles (squeeze points) often precede significant price movements or trends in the cumulative returns.

The bottom chart shows the ratio of the Bollinger Bands width to the Keltner Channel width (BB/KC Width Ratio), which are measures of volatility. The red dashed line represents the squeeze threshold (e.g., when the ratio is below 1, a squeeze occurs). Notice that when the ratio goes below zero, a squeeze event occurs in the top chart.

Simple Moving Average

A Simple Moving Average (SMA) is a technical indicator that calculates the average price of an asset over a set period, smoothing out short-term fluctuations to reveal trends. Short-term SMAs (e.g., 50 periods) respond quickly to price changes and capture recent momentum, while long-term SMAs (e.g., 200 periods) provide a broader view of the overall market trend.

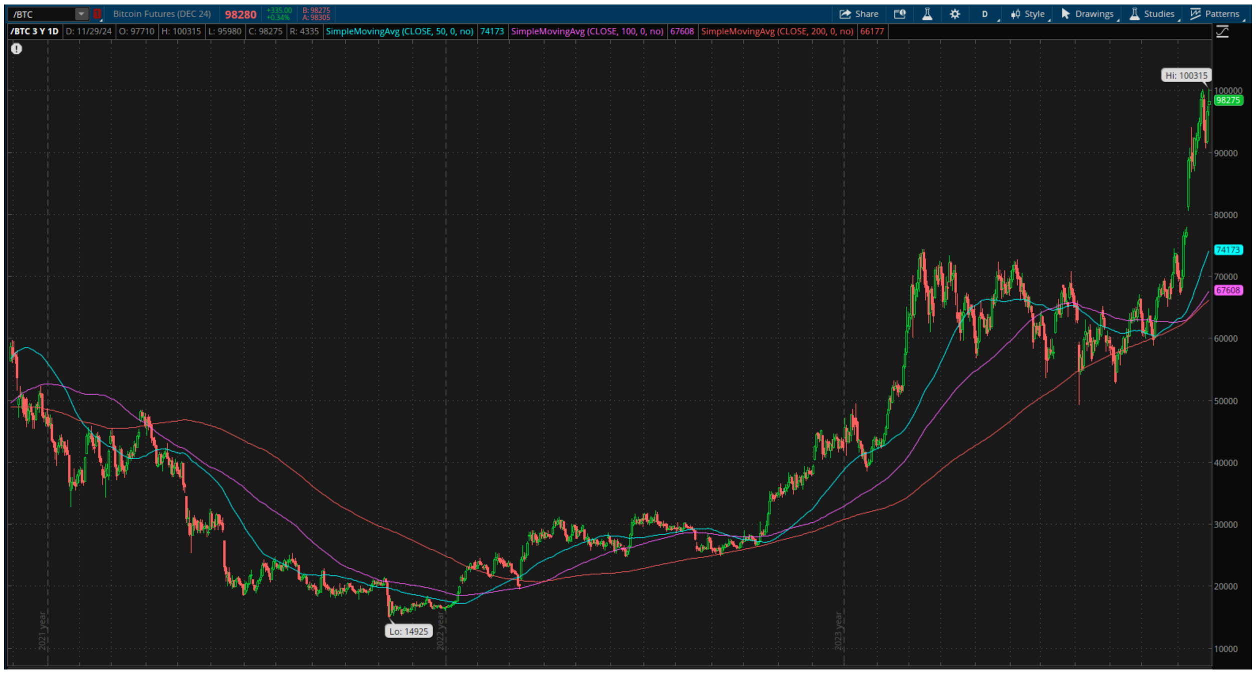


Figure 3: Bitcoin Simple Moving Average. ThinkOrSwim (2024).

The image explains Simple Moving Averages (SMAs), which smooth out price fluctuations to highlight trends.

- Blue (short-term SMA 50): Tracks recent trends.
- Purple (medium-term SMA 100): Balances short- and long-term signals.
- Red (long-term SMA 200): Shows overall market direction.

Exponential Moving Average

An Exponential Moving Average (EMA) is a type of moving average that gives more weight to recent prices, making it more responsive to recent market changes compared to a Simple Moving Average (SMA). The EMA is calculated by applying a smoothing factor to prioritize recent data, which helps traders identify trends, reversals, and momentum more quickly. It is often used in conjunction with other EMAs (e.g., 12-period and 26-period) to generate buy or sell signals when crossovers occur, making it particularly useful in dynamic markets.

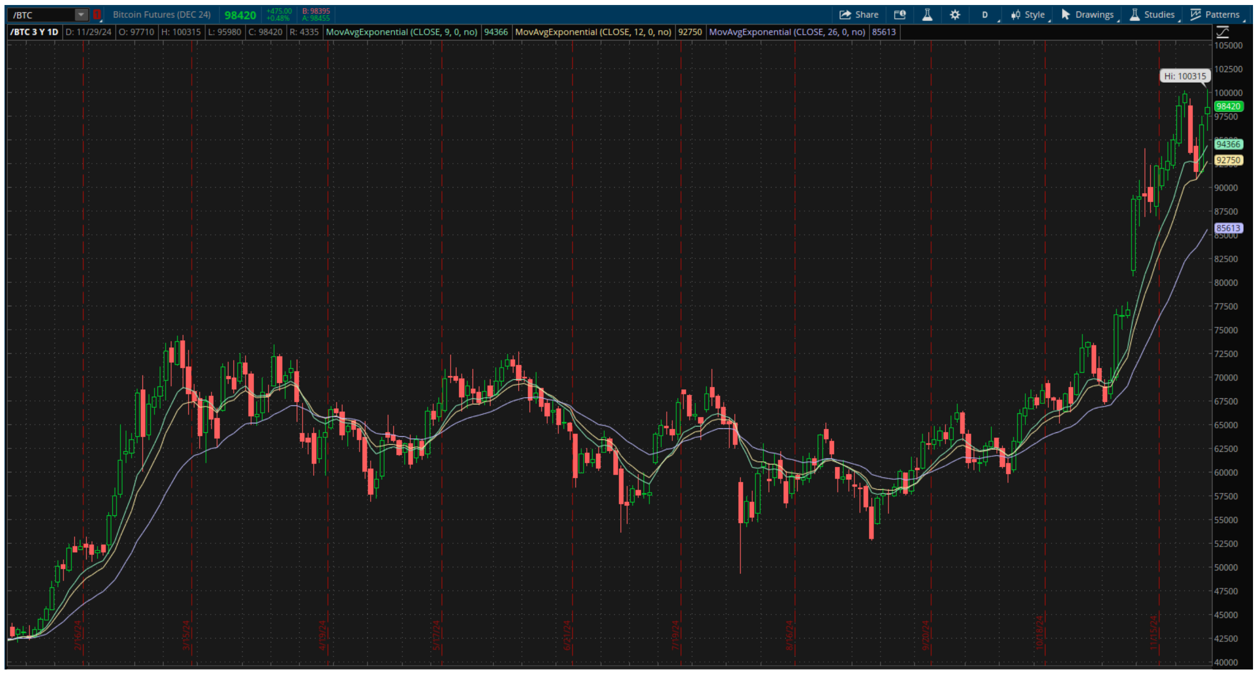


Figure 4: Bitcoin Exponential Moving Average. ThinkOrSwim (2024).

The image explains Exponential Moving Averages (EMAs), which emphasize recent price changes to highlight trends.

- Green (short-term EMA 9): Tracks immediate price momentum.
- Yellow (medium-term EMA 12): Balances short-term responsiveness and stability.
- Blue (long-term EMA 26): Highlights broader market trends.

A key phenomenon to highlight with EMA, are **EMA Crosses**. EMA Crosses occur when two Exponential Moving Averages (EMAs) with different time periods intersect, signaling potential trend shifts:

- Bullish Crossover (Golden Cross): The shorter EMA (e.g., 9-day) crosses above the longer EMA (e.g., 26-day), indicating upward momentum and a potential buy signal.
- Bearish Crossover (Death Cross): The shorter EMA crosses below the longer EMA, signaling downward momentum and a potential sell signal.

EMA crosses are more sensitive to recent price changes, making them effective for identifying early trend reversals in dynamic markets.

Sources & Methods of Data Acquisition

To capture these key stock metrics, we collected data from several sources.

The primary data sources are:

1. **Yahoo Finance (yfinance):** Provides historical Bitcoin price data and financial indicators essential for trend analysis. We utilized standard financial metrics such as high, low, open, close, date, and volume.
2. **GDELT:** A global database that aggregates worldwide news every 15 minutes. Offers social sentiment data to gauge market perception.

Methods of Acquisition

- Data from yfinance is retrieved via API, focusing on daily price and volume data.
- GDELT data is scraped and analyzed for sentiment using natural language processing tools to extract features.

Data Pre-Processing & Exploration

Feature Engineering & Feature Selection

Stock returns frequently show excess kurtosis, meaning they have fatter tails and a sharper peak than the normal distribution, which is referred to as "leptokurtic." This characteristic indicates a higher probability of extreme returns (both high and low) compared to what the normal distribution would predict.

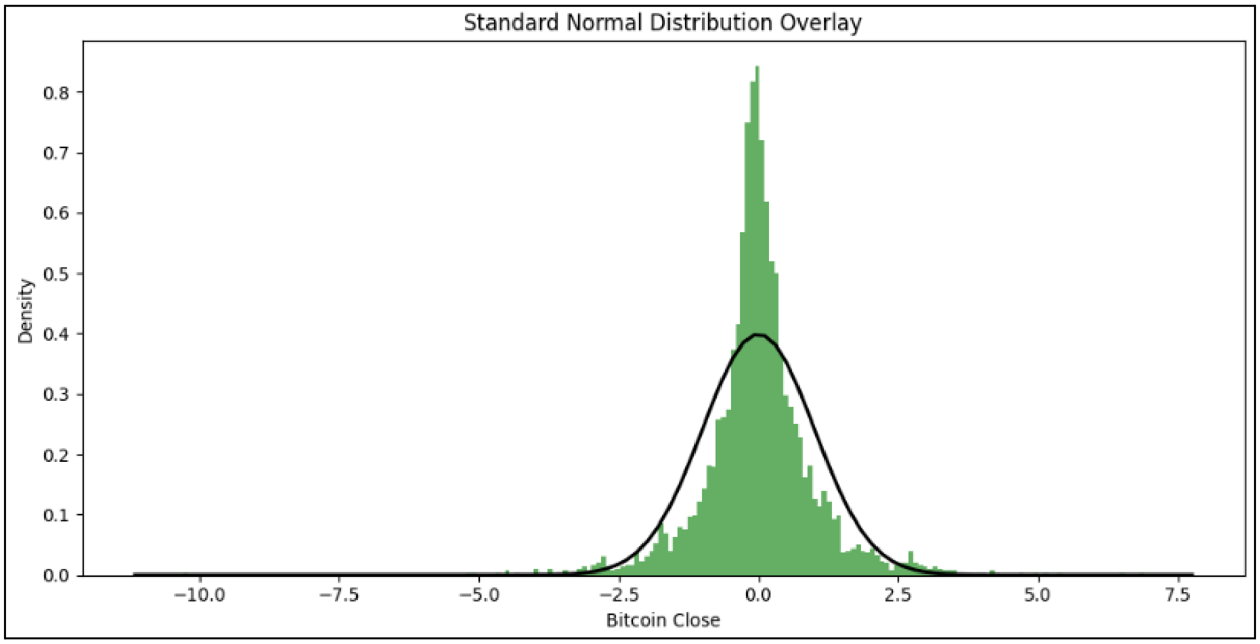


Figure 5: Distribution of Bitcoin (green) compared to Standard Normal Distribution (black).

Reddit Scraping and Sentiment Analysis

Initially, we intended to scrape data from Reddit to match every day of the Bitcoin price data we had obtained from yfinance, from 2014 to 2024. However, attempting to scrape with Reddit's API led to limited data with many missing values and inconsistent posts from the subreddits, as illustrated in *Figure 6*. This limited the impact and applicability of our sentiment analysis.

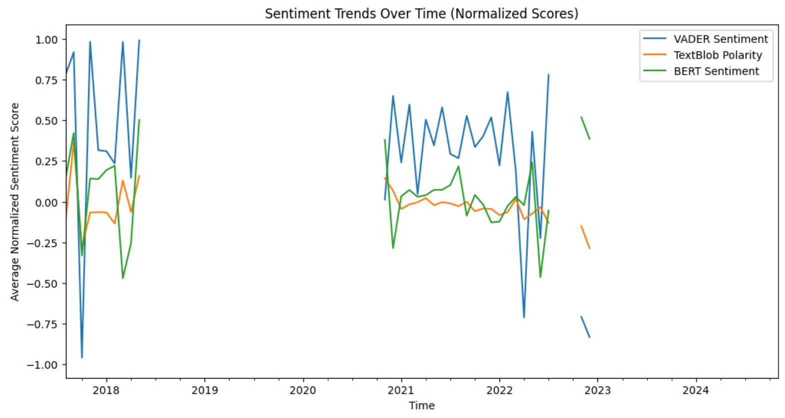


Figure 6: Sentiment Trends Over Time Using Reddit Data

Additionally, sentiment analysis with VADER and TextBlob as opposed to BERT revealed differing scores for each type of analysis among subreddits, implying that there could be nuances in the reddit content or sarcastic posts that only an LLM like BERT was able to catch.

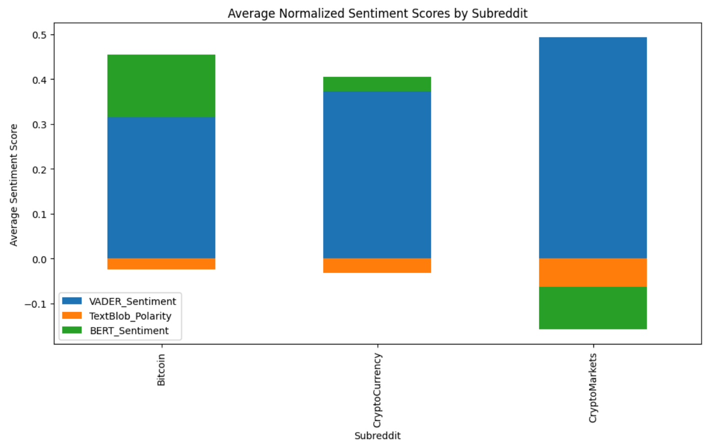


Figure 7: Average Normalized VADER, TextBlob, and BERT Sentiment Scores Across Subreddits

This was confirmed when we repeated the same process for 1 year of price data, from December 2023 - December 2024, scraped from Coingecko (a popular Cryptocurrency news platform) and tried to match that with Reddit data scraped from the same time period. We were able to get 164 clean values out of 365 days, with limited predictive insights. However, since this performed better than the results obtained with yfinance, we proceeded with visualization and analysis of the relationship between sentiment and price.

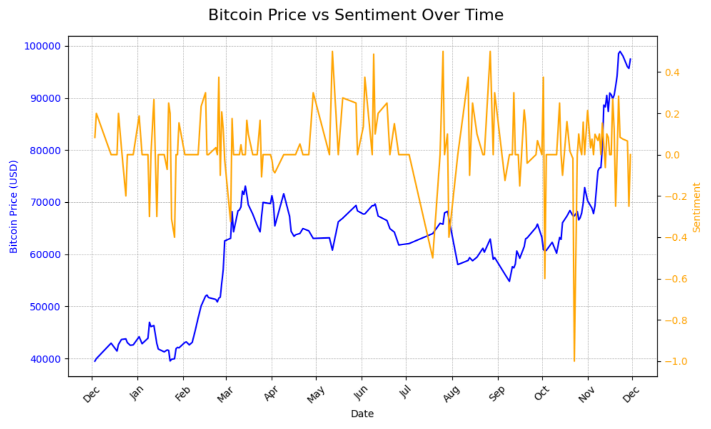


Figure 8: Plot of Bitcoin Price vs. Sentiment for the Last 1 Year Using Coingecko

The results showed little long-term relationship between sentiment and bitcoin prices, and sentiment and price were both volatile with unstable alignment over time. However, a lagged correlation analysis shows that sentiment had the potential to impact price over a rapid time frame, peaking at 3 days. This indicates that sentiment from 3 days before a price change had some ability to predict that price change.

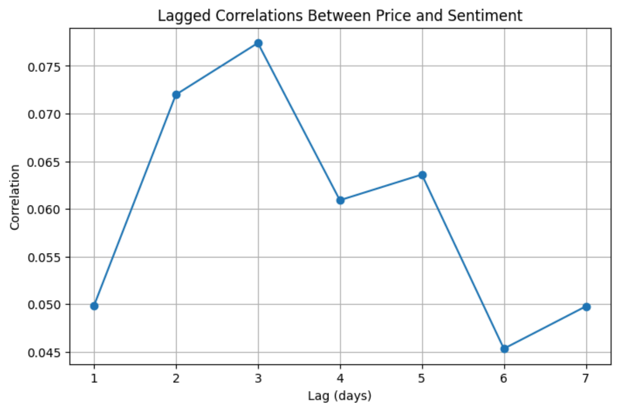


Figure 9: Lagged Correlations Between Bitcoin Price and Reddit Sentiment

At this point, since sentiment analysis with Reddit data was not providing impactful results, we pivoted to data from the GDELT project, a global database that aggregates worldwide news every 15 minutes and assigns its own sentiment scores to those articles. Using Google BigQuery's API, we obtained entries containing keywords such as “bitcoin”, “blockchain”, and “crypto” for every day from 2014 to 2024. Then, to confirm the validity of the provided sentiment scores, we went back to the source news URLs, tokenized and translated the article titles to English (since processing the full articles was too time- and memory-intensive for the purposes of the project), and performed sentiment analysis with VADER and BERT to cross-check.

Ultimately, GDELT's provided average sentiment score proved to be the best reflection of the sentiment from the news articles and was used for further analyses.

Now that we had reliable sentiment data for 2014-2024, we merged this with our previous yfinance data to analyze the relationship between sentiment and price. Again, even with 10 years of data, there was minimal relation between sentiment and bitcoin prices, confirming that human sentiment or news articles are not reliable predictors of cryptocurrency prices.

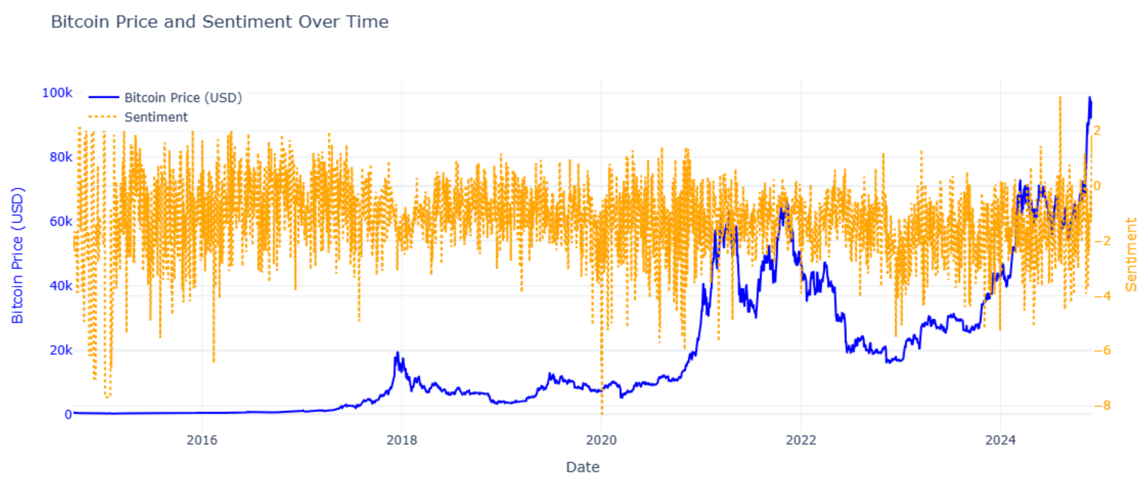


Figure 10: Plot of Bitcoin Price and GDELT Sentiment from 2014-2024

The only point at which sentiment data coincided with price trends was in 2021, which points to the possibility of the COVID-19 pandemic having some impact on the popularity of cryptocurrency and people's awareness of it. However, in the long term, no measure of sentiment was related to price. In fact, dividing sentiment into polarity categories of positive, neutral, and negative actually suggested that days with overwhelmingly negative sentiment were related to higher prices, which is the opposite of what we had expected.

Looking into this further, there seemed to be a predictable relationship between price and sentiment for days with extreme sentiment. We normalized all our sentiment scores to range from -1 to 1, so this meant that on days with extremely positive (greater than 0.8) or extremely negative (less than -0.8) sentiment, sentiment actually did have some ability to predict price. This implies the possibility of anomaly-driven trading if our model was able to predict just these extreme sentiment days. However, since there are only a few such days, we will have to consider how useful such a modification would be compared to the potential returns of correctly predicting an extreme sentiment day.

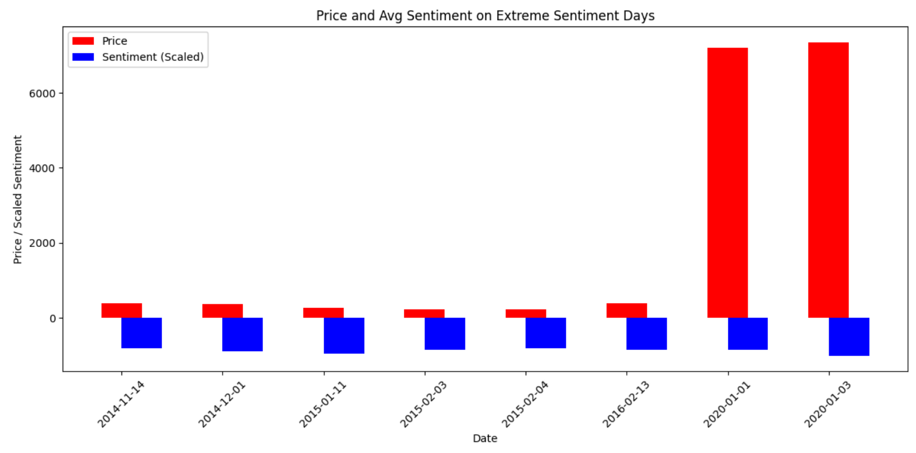


Figure 11: Price and Average Scaled Sentiment on Extreme Sentiment Days

In the future, we will need to improve upon this by testing nonlinear models for the relationship between sentiment and price, as well as diving deeper into the impact of the COVID-19 pandemic and extreme sentiment days. However, for the time being, we have incorporated GDELT sentiment as a parameter in the existing model to better inform price predictions.

Input Variables

Daily Prices: Open, High, Low, and Close Price
Sentiment Score: GDELT Average Sentiment
Squeeze: Binary Momentum Predictor
EMA_9: Exponential Moving Average across 9 periods

- Also EMA_12 & EMA_26

SMA_50: Simple Moving Average across 50 periods

- Also SMA_100 & SMA_500

Cross 9_12_UP: Binary indicator when EMA 9 crosses above EMA 12

- Also Cross 9_26_UP & 12_26_UP

Cross 9_12_DOWN: Binary indicator when EMA 9 crosses below EMA 12

- Also Cross 9_26_DOWN & 12_26_DOWN

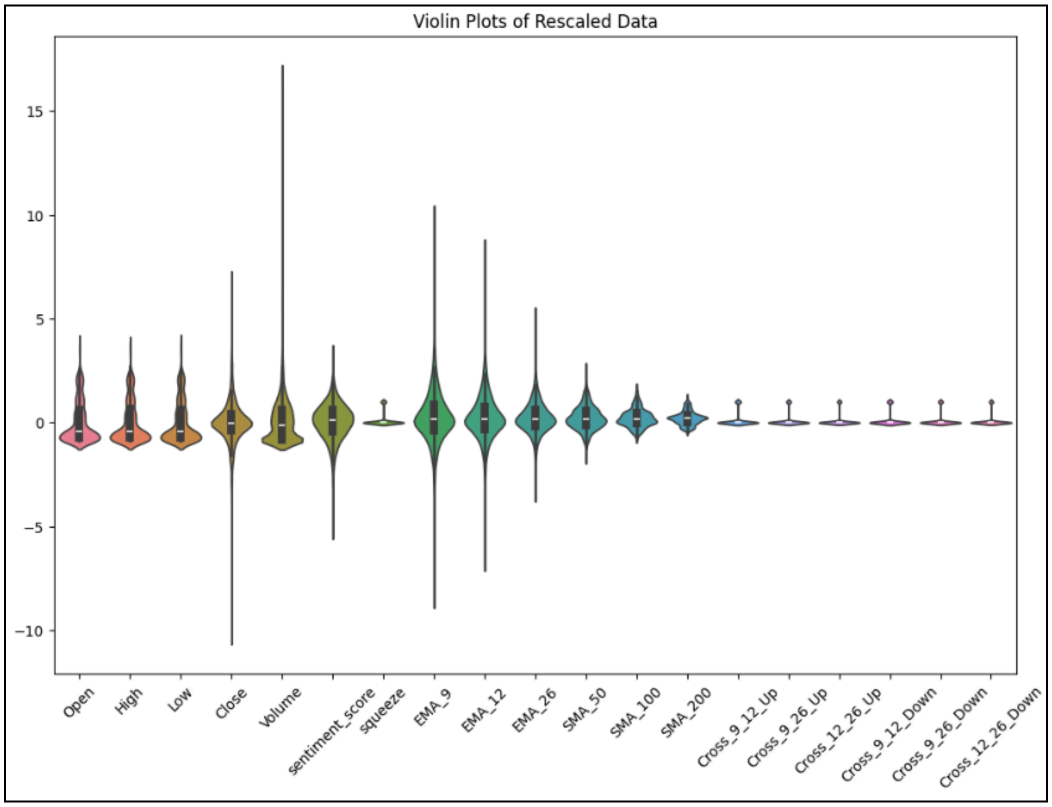


Figure 12: Violin Plot of Data Rescaled Using Z-Score

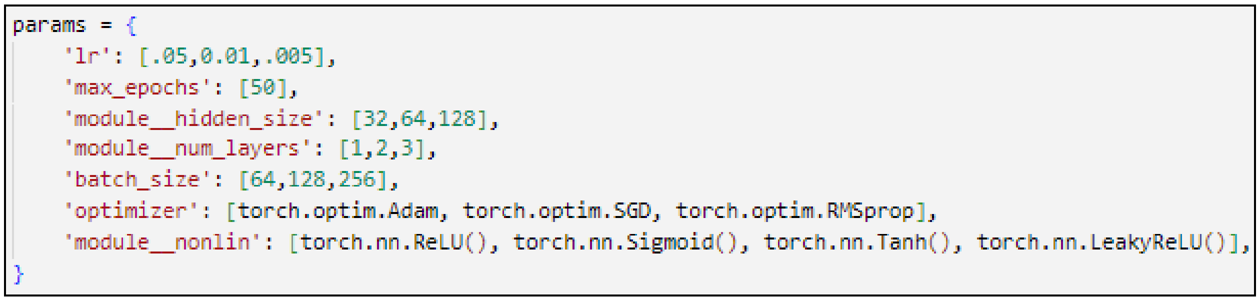
Methods

1. Initial Model

The initial base model was designed to predict Bitcoin prices directly using historical data and basic financial indicators. This model was intended as a starting point to explore the common misconceptions and pitfalls of predicting the market.

For this model, we implemented a Long Short-Term Memory (LSTM) network. This is a specialized type of recurrent neural network (RNN) designed to model sequential and time-series data effectively. The LSTM architecture was chosen due to its ability to capture temporal dependencies and handle the inherent challenges of financial data, such as non-linearity, volatility, and long-term patterns.

Furthermore, we leveraged GridSearch Cross-Validation (CV) to identify the optimal hyperparameters for the model by tuning the following parameters:



```
params = {
    'lr': [.05,0.01,.005],
    'max_epochs': [50],
    'module__hidden_size': [32,64,128],
    'module__num_layers': [1,2,3],
    'batch_size': [64,128,256],
    'optimizer': [torch.optim.Adam, torch.optim.SGD, torch.optim.RMSprop],
    'module__nonlin': [torch.nn.ReLU(), torch.nn.Sigmoid(), torch.nn.Tanh(), torch.nn.LeakyReLU()],
}
```

Figure 13: Parameters used as input for GridSearch

The initial base model demonstrated promising performance, as reflected in its evaluation metrics and visualization of predictions. The results are summarized below:

Results

- **Test Mean Squared Error (MSE):** 0.03
- **Test R-Squared (R²):** 0.97

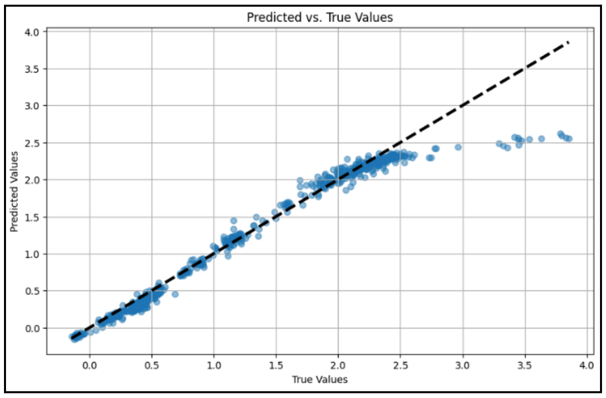


Figure 14: Visualization of Predicted Vs Actual Values

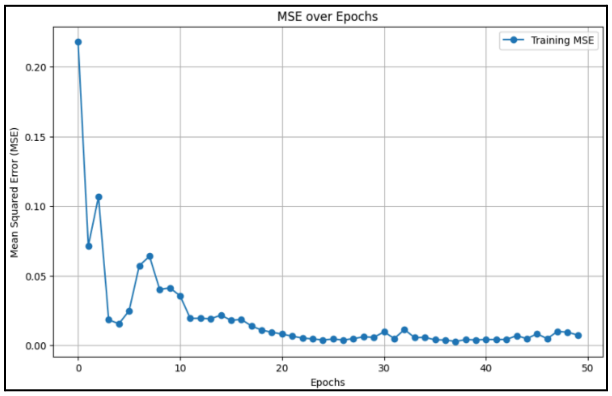


Figure 15: Visualization of MSE after each epoch

The scatter plot on the left shows a strong linear relationship between predicted and actual values, with most points aligning closely along the diagonal line (representing accurate predictions). Similarly, the training MSE plot on the right highlights a rapid decline in error during the initial epochs, followed by stabilization as the model converges, suggesting effective learning and optimization.

While these results appear promising at first glance, they highlight a fundamental issue often encountered in this type of prediction task. Since cryptocurrency prices tend to exhibit relatively small day-to-day changes and remain near the same value, models can achieve seemingly high accuracy by simply predicting values close to the previous day's price. However, this approach is misleading and fails to capture the true goal of the analysis.

The key objective is not merely to predict the next day's price but to understand and forecast the percentage increase or decrease in value. This shift in focus is essential for generating actionable insights, particularly in financial trading, where percentage changes drive decision-making rather than absolute price levels. By predicting percentage returns instead of prices, we account for the dynamic nature of market movements and ensure that the model captures meaningful trends rather than trivial patterns. This realization highlights the importance of redefining the target variable to align with the practical goals of the project and improve the model's usability in real-world scenarios.

2. Corrected Base Model, Expanded Methodology

Based on our learnings from the previous section, we made two key changes to our base model:

1) Use daily returns in our model instead of daily prices
2) Use a new hyperparameter tuning technique - HyperOpt

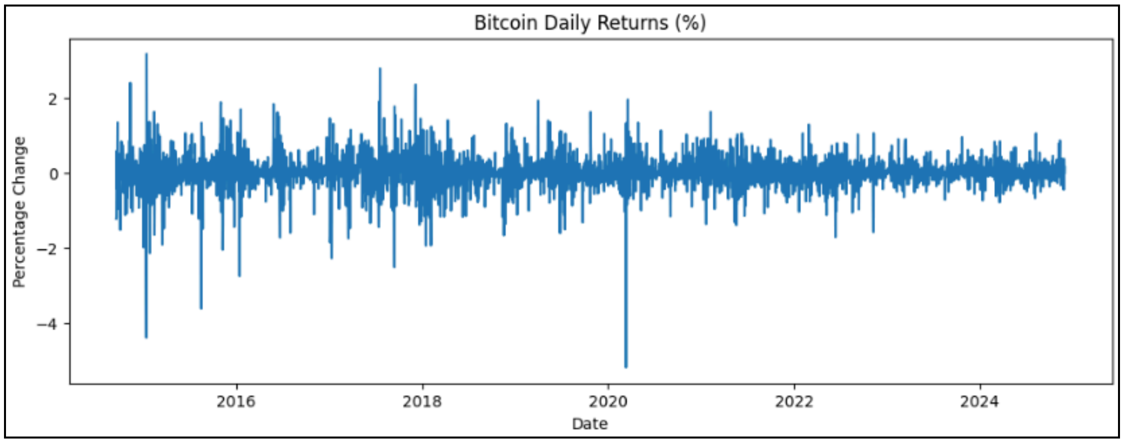


Figure 16: Plot of Bitcoin Daily Returns (Percentage Change) Over Time

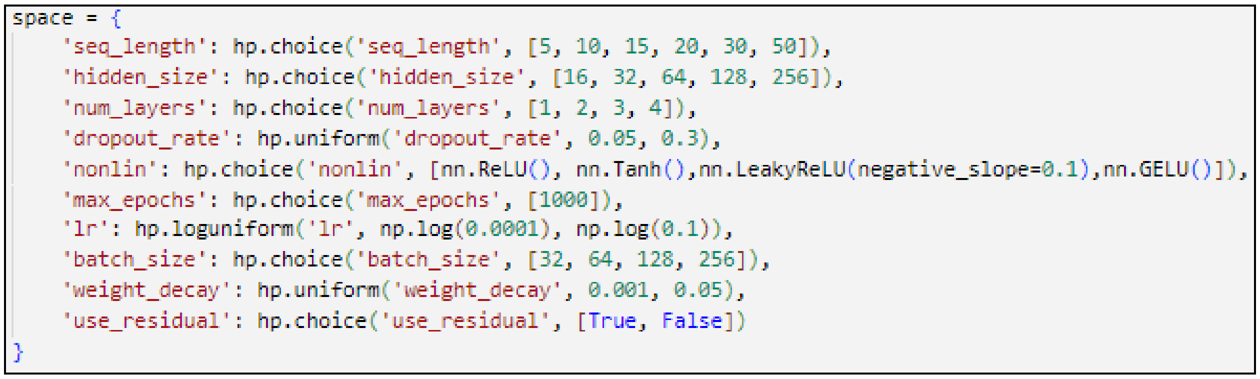
HyperOpt Parameter Tuning

With our new and improved Daily Returns as our input, we decided to switch from GridSearchCV to HyperOpt for faster and more efficient hyperparameter tuning, especially for large and complex models. Instead of testing every possible combination, Hyperopt uses past results to focus on the most promising options, saving time and computation. It also handles advanced settings, like sequence length (data input size) and dropout rates (neuron deactivation), more easily. This smarter approach helps find better hyperparameters without the heavy cost of exhaustive searches, making it ideal for tuning deep learning models and other complex tasks.

Specifically, in our hyperparameter tuning we used a set of Training Callbacks to fine tune our model each epoch: Loss Callback, Gradient Clipping, Learning Rate Scheduler, and Early Stopping. During training, we set up the space such that the Loss Callback monitors the loss at each epoch to detect plateaus, such as when the loss stalls at 0.8 for multiple epochs. The Learning Rate Scheduler (ReduceLROnPlateau) responds by halving the learning rate up to three times during the plateau, enabling the optimizer to make finer adjustments. If no further improvement is achieved after these reductions, Early Stopping terminates the training process to prevent overfitting and conserve computational resources.

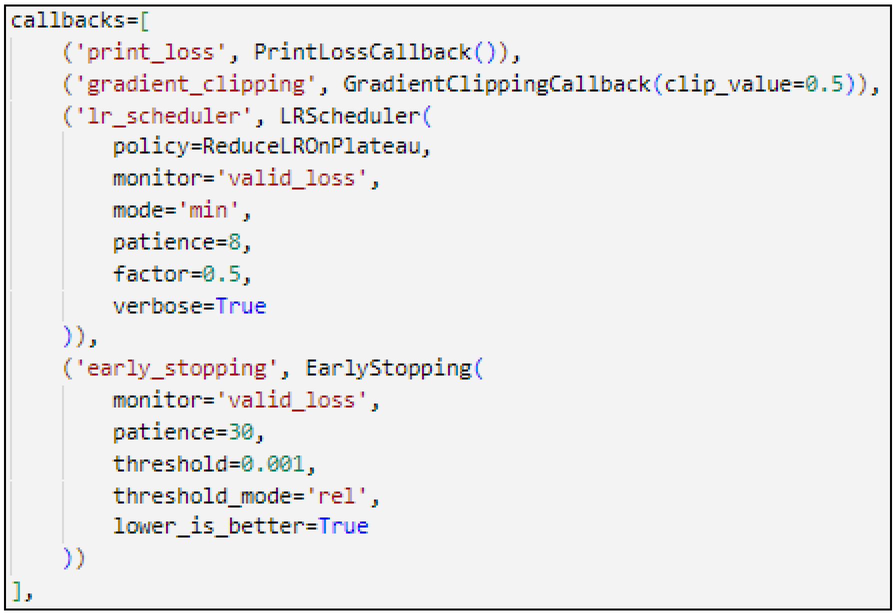
Clarifying Definitions:

- *Loss Callback: Specifically tracks and logs loss values for manual evaluation, helping assess performance in real time.*
- *Gradient Clipping: Prevents exploding gradients by capping gradient values during backpropagation.*
- *Learning Rate Scheduler: Dynamically adjusts the learning rate based on validation loss trends using ReduceLROnPlateau.*
- *Early Stopping: Stops training when validation loss stops improving, avoiding overfitting and saving computation.*



```python
space = {
    'seq_length': hp.choice('seq_length', [5, 10, 15, 20, 30, 50]),
    'hidden_size': hp.choice('hidden_size', [16, 32, 64, 128, 256]),
    'num_layers': hp.choice('num_layers', [1, 2, 3, 4]),
    'dropout_rate': hp.uniform('dropout_rate', 0.05, 0.3),
    'nonlin': hp.choice('nonlin', [nn.ReLU(), nn.Tanh(),nn.LeakyReLU(negative_slope=0.1),nn.GELU()]),
    'max_epochs': hp.choice('max_epochs', [1000]),
    'lr': hp.loguniform('lr', np.log(0.0001), np.log(0.1)),
    'batch_size': hp.choice('batch_size', [32, 64, 128, 256]),
    'weight_decay': hp.uniform('weight_decay', 0.001, 0.05),
    'use_residual': hp.choice('use_residual', [True, False])
}
```

Figure 17: HyperOpt Parameter Tuning Space Snapshot



```
callbacks=[
    ('print_loss', PrintLossCallback()),
    ('gradient_clipping', GradientClippingCallback(clip_value=0.5)),
    ('lr_scheduler', LRScheduler(
        policy=ReduceLROnPlateau,
        monitor='valid_loss',
        mode='min',
        patience=8,
        factor=0.5,
        verbose=True
    )),
    ('early_stopping', EarlyStopping(
        monitor='valid_loss',
        patience=30,
        threshold=0.001,
        threshold_mode='rel',
        lower_is_better=True
    ))
],
```

Figure 18: Model Training Callbacks Set Up Snapshot

Results

- **Test Mean Squared Error (MSE):** 0.51
- **Test R-Squared (R^2):** 0.00

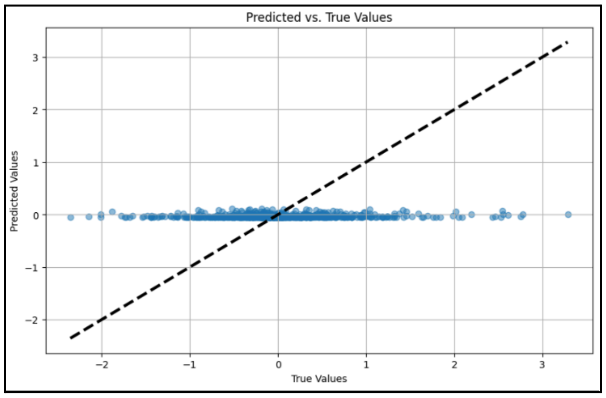


Figure 19: Visualization of Predicted Vs Actual Values

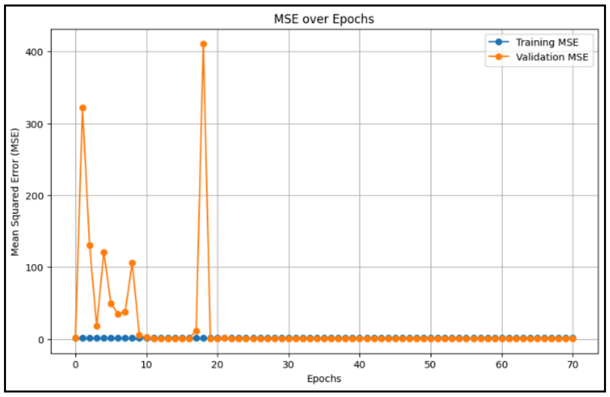


Figure 20: Visualization of MSE after each epoch

Our model's performance worsened when predicting returns instead of prices, which is an expected tradeoff for generating more actionable insights. While predicting prices often achieves lower MSE and higher R-squared due to the model simply approximating recent values, this approach lacks meaningful predictive power. In contrast, predicting returns introduces greater variability and complexity, as returns are more volatile and harder to model. This led the model to converge around 0, minimizing error but offering little predictive value. This shift highlights the challenges of capturing meaningful patterns in return data compared to the smoother trends seen in price predictions.

3. Classification

At this point, we took another approach where we switched from a regression model predicting the percentage change in price per day to a binary classification model, with positive percent change being 1 and a negative percent change classified as 0. However, our classification model predicted all positives and converged rapidly.

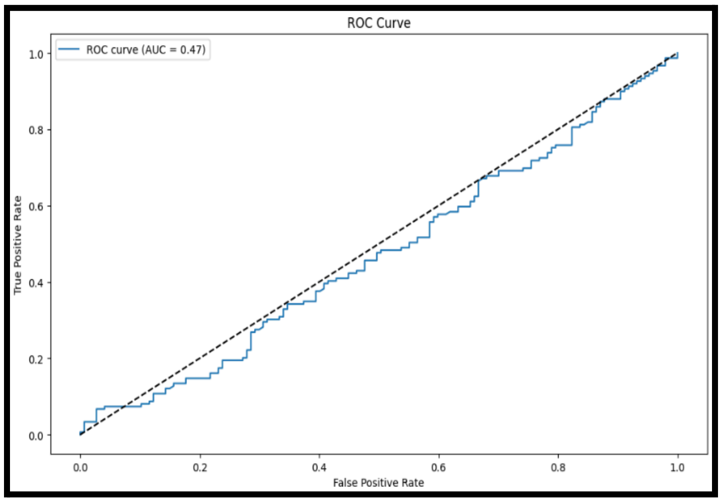


Figure 21: ROC Curve for Classification Model

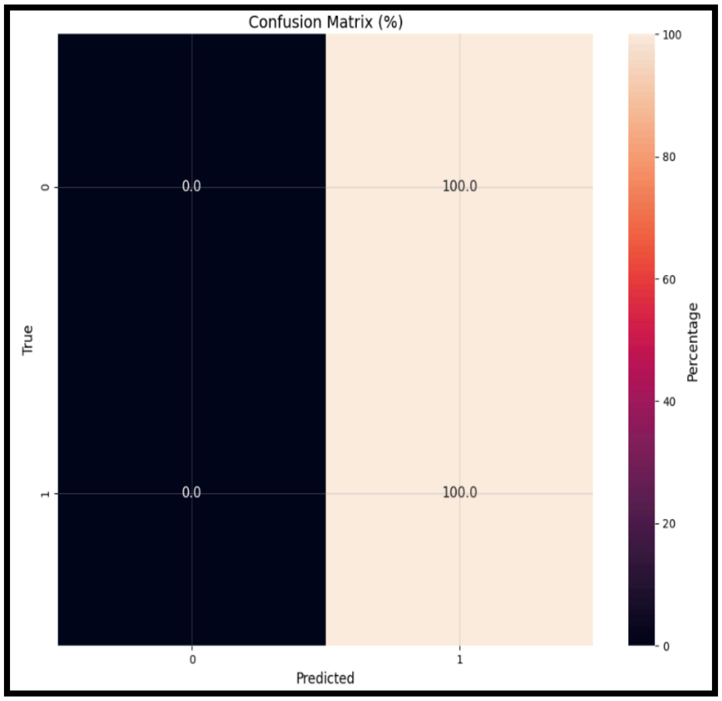


Figure 22: Confusion Matrix for Classification Model that Shows all Positive Predictions

To address these limitations caused by trivial predictions, we built and incorporated a custom loss function called NoSmallLoss.

4. NoSmallLoss

Key challenges we faced from our previous models are the tendency to predict trivial values near zero and poorly performing binary classification. While this minimized the loss, it failed to capture meaningful patterns of the data. For instance, when predicting percentage changes in prices, the model consistently predicted values between -0.1 and 0.1, which reduced the mean squared error (MSE) but failed to capture the directional movements in the data. To overcome this limitation, we needed a targeted solution that could force the model to go beyond these trivial predictions while aligning the true distribution of returns.

Custom Loss Function

To address this issue, we developed a custom loss function called NoSmallLoss, designed to penalize predictions within a specified range and reward predictions that align more closely with the target distribution. Through this approach, the model generated more meaningful and actionable outputs.

NoSmallLoss operates based on three key parameters:

1. loss_lower_bound and loss_upper_bound: These two parameters define the range of predictions to penalize (e.g., -0.1 to 0.1). Predictions within this range are penalized more heavily than others, discouraging the model from defaulting to "safe" outputs.
2. penalty_weight: This determines the severity of the penalty applied to predictions. A higher weight forces the model to avoid the penalized range more aggressively.

By incorporating these parameters into the model, NoSmallLoss dynamically adjusts the learning process. It introduced a "bias" against trivial outputs, encouraging the model to learn patterns that are more aligned with true market behavior.

Results

The NoSmallLoss function improved the model's performance in generating more realistic predictions. Below are some key improvements we observed, supported by visuals:

1. Improved alignment with true values

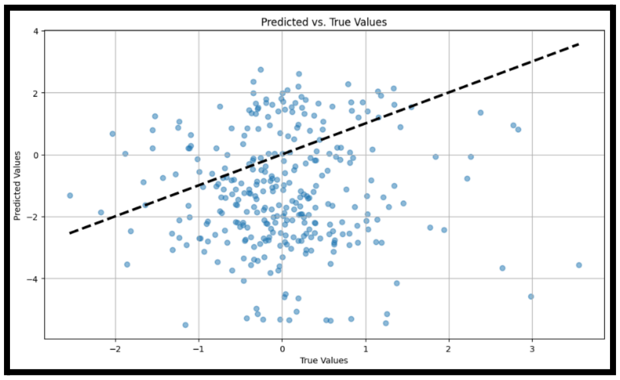


Figure 23: Visualization of Predicted vs Actual Values

The scatterplot above shows how NoSmallLoss transformed the model's predictions. Predictions are now more closely aligned with the true values (dashed lines), compared to the earlier corrected base model trivial outputs clustered near zero.

2. Enhanced Distribution Alignment

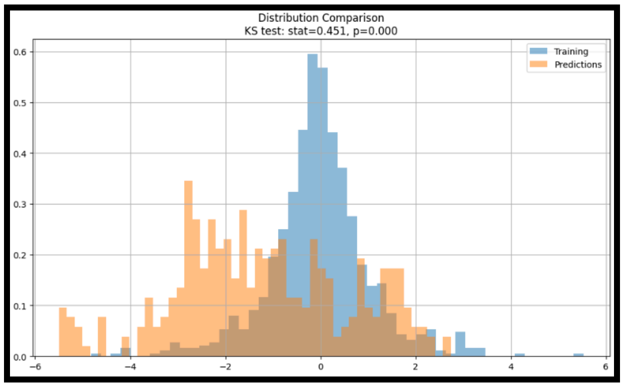


Figure 24: Distribution of Training vs Predicted with KS test

Using the Kolmogorov-Smirnov (KS) test, we evaluated the alignment of the predicted values with the true data distribution. The histogram above demonstrates that the predicted values (orange) are now much closer to the training data (blue), indicating a statistically significant improvement.

3. Reduction in MSE over time

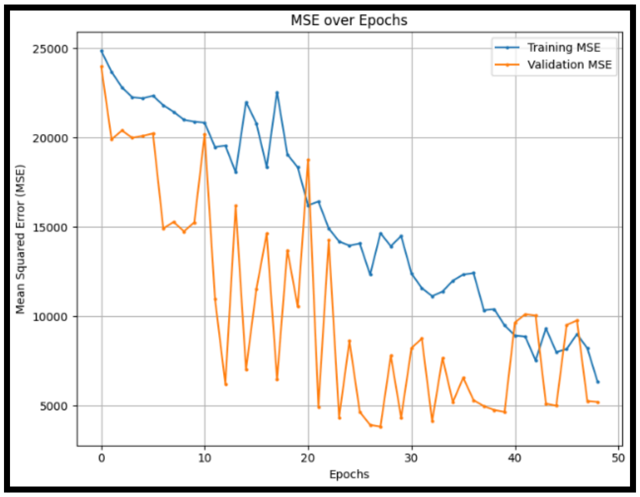


Figure 25: MSE over epochs

The MSE over trials of epochs highlights the gradual improvement in validation error as the model adapted to the NoSmallLoss. While the overall MSE is slightly higher than before due to the penalization of trivial outputs, this tradeoff resulted in higher quality predictions that better reflect true market behavior in return.

Tradeoff: The Need for Pruning

While NoSmallLoss reduced trivial predictions, it introduced a tradeoff: an increase in MSE and computational overhead.

1. Higher MSE: The model was no longer relying on "safe" outputs near zero to minimize loss. Instead, it was encouraged to explore a broader range of predictions, which generated a higher margin of error.
2. Computational Inefficiency: Training neural networks often involves running multiple trials with different hyperparameters, many of which result in poor performing models. These trials consume valuable computational resources, especially when training on large datasets or complex models like LSTMs. In our case, poorly performing trials often stagnated at high loss values, contributing little to the optimization process but requiring significant computational time.

To address this, we implemented dynamic pruning.

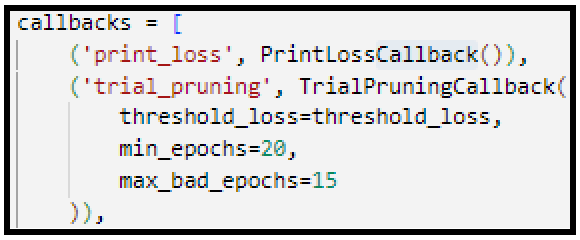
Dynamic Pruning

Dynamic pruning is similar to early stopping, but instead of stopping training for a single model, it operates at the trial level during hyperparameter optimization. By identifying and stopping unpromising trials early, this approach significantly reduces computational time while maintaining the quality of the optimization process.

Dynamic pruning operates based on three parameters:

1. min_epochs=20: This ensures that every trial runs for at least 10 epochs before evaluation, allowing the model enough time to establish the trajectory. Most failing trials showed poor performance within the first 5 epochs.
2. max_bad_spochs=15: This stops trials that show no improvement for 5 consecutive epochs, preventing computation on stagnant models
3. threshold_mse: Initially set to 30,000, the threshold is dynamically adjusted throughout training based on 100x the best loss seen so far, with a cap at 20.0. This ensures poor trials (MSE >10-20) are pruned early, while promising ones continue with a progressively narrowing threshold.

These parameters were implemented using callbacks, as shown below, allowing us to automate the pruning process.



```
callbacks = [
    ('print_loss', PrintLossCallback()),
    ('trial_pruning', TrialPruningCallback(
        threshold_loss=threshold_loss,
        min_epochs=20,
        max_bad_epochs=15
    )),
```

Figure 26: Implementation of dynamic pruning using callbacks

Impact of Pruning

The impact of dynamic pruning was key, enabling us to optimize NoSmallLoss and other hyperparameters with greater efficiency. Below are some key results:

1. Reduced Computation Overhead

The top graph (*Figure 27)* below shows the hyperparameter optimization progress. Spikes in trial loss indicate poor performing trials that were terminated early, allowing the optimization process to focus on better performing trials.

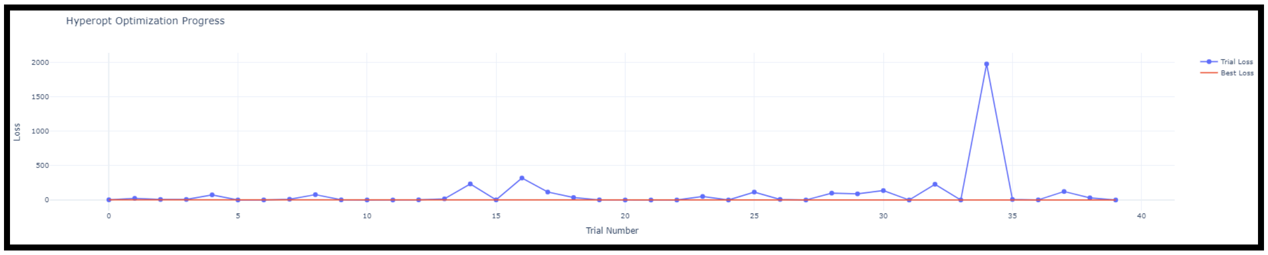


Figure 27: HyperOpt Loss by Trial

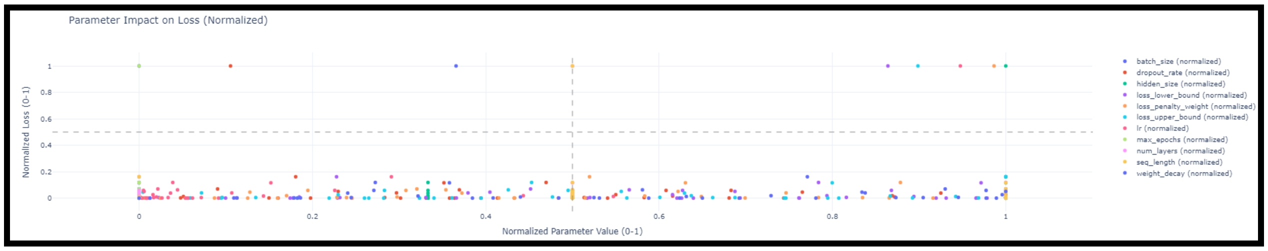


Figure 28: Hyperparameter Impact on Loss

2. Improved Parameter Focus

The bottom graph (*Figure 28)* above highlights the impact of various hyperparameters on the loss. Dynamic Pruning allowed us to prioritize impactful parameters, such as threshold_mse, and discard those with minimal contribution.

By combining NoSmallLoss and dynamic pruning, we were able to enhance not only the quality but also the efficiency of our predictive model. NoSmallLoss ensured that the model generated meaningful predictions aligned with the true data distribution, while dynamic pruning streamlined the optimization process, making the project computationally feasible without compromising on performance.

Model Robustness

One of the key strengths of this project lies in its flexibility and robustness, allowing the model to be applied to any ticker or timeframe beyond just Bitcoin at a daily rate. This adaptability was demonstrated by testing the model on NVDA (NVIDIA) stock at an hourly rate, showcasing its ability to handle different datasets and granularities of time. The results of the model are shown below:

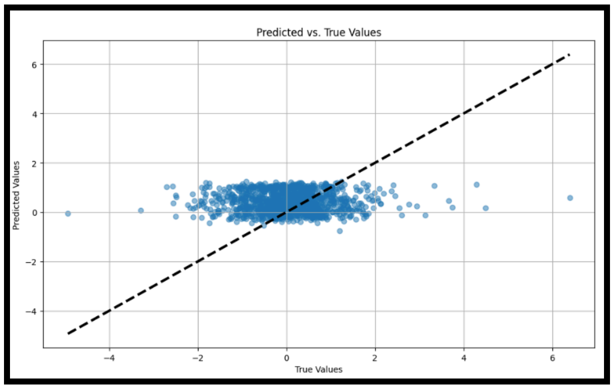


Figure 29: Visualization of Predicted vs Actual Values

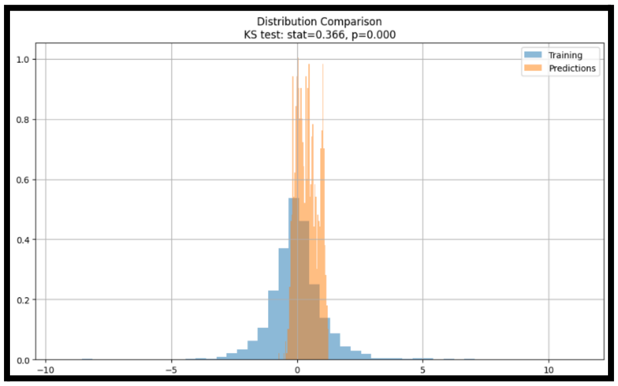


Figure 30: Distribution of Training vs Predicted with KS test

Results & Insights

We developed five models to tackle the complex problem of predicting financial asset prices. Each model offered unique insights, highlighting the intricate nature of financial markets. Here, we detail the performance and findings of each model without overloading the section with graphics, focusing instead on comparing the distribution of results across models.

Base Bi-directional LSTM Model

Our initial model utilized the true price as the target variable, achieving a test Mean Squared Error (MSE) of 0.03 and an R-squared of 0.97. However, these impressive metrics underscore a critical flaw—relying on the true price of a non-stationary financial asset tends to predict prices extremely close to the most recent, reflecting little of the broader price distribution. *Refer to Figure #14.*

Expanded Model with Hyperparameter Optimization

We refined our approach by logarithmically transforming the price data and targeting percentage changes as returns. After initial poor performance, we expanded our hyperparameter space to include key LSTM parameters such as dropout rate and sequence length. Utilizing a more sophisticated tuning algorithm, HyperOpt (as opposed to the traditional GridSearchCV), the model predominantly predicted values close to zero, with an MSE of 0.51 and an R-squared of 0. This illustrates the profound difficulty in predicting financial returns even with advanced methodologies. *Refer to Figure #19.*

Classification Model

Adapting the LSTM model to a binary classification task, we aimed to predict daily price movements as either positive (1) or negative (0). Unsurprisingly, as the previous model might suggest, this model also underfits, predicting a positive return for all inputs, resulting in a Receiver Operating Characteristic Area Under the Curve (ROC-AUC) of 0.5—indicative of no predictive capability beyond random chance. *Refer to Figure #21.*

Model with Custom Loss Function

To address the tendency of our previous underfitting corrected regression model to predict near-zero values, we introduced a custom loss function, incorporating parameters for loss boundaries and a penalty weight. This approach succeeded in producing predictions that better mirrored the actual distribution of the target variable. However, this success sacrificed accuracy, as demonstrated by an MSE of 5.5 and an R-squared of -6.99 in our most extensive test. *Refer to Figures #23 & #24.*

Robust Model for Various Assets and Timeframes

Finally, we broadened our scope to predict various financial assets across different time frames. For example, applying our adjusted model to predict hourly price movements of Nvidia (NVDA) resulted in an MSE of 1.11 and an R-squared of -0.37. While these results were more promising compared to our daily Bitcoin predictions, they highlight the substantial computational demands of optimizing hyperparameters—a task that may benefit from cloud computing and alternative tuning strategies in future research.

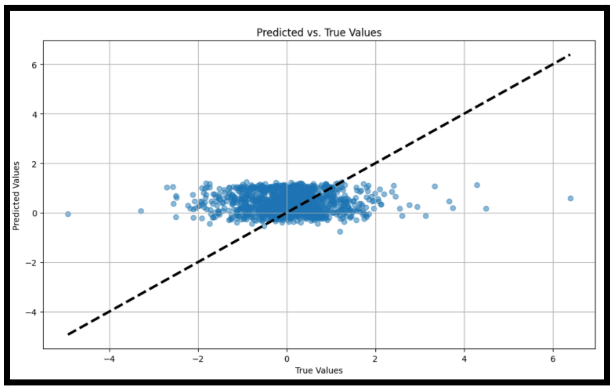


Figure 31: Visualization of Predicted vs Actual Values

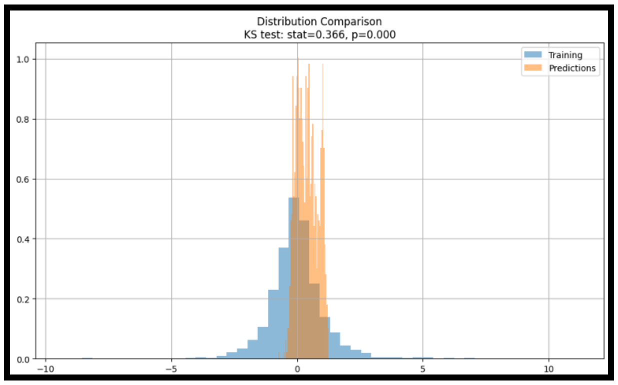


Figure 32: Distribution of Training vs Predicted with KS test

Conclusion

The project demonstrated that while custom loss functions can align model predictions with realistic market behaviors, this often comes at the cost of decreased predictive accuracy. The robustness of our models across different financial assets and timeframes highlighted their adaptability. We also established the importance of scalable methodologies, utilizing advanced optimization and regularization techniques like HyperOpt and dropout. A key insight from our experiments was the necessity of predicting percentage returns instead of absolute prices to effectively manage the non-stationarity in financial data.

We learned that increasing the complexity of models can lead to a trade-off between adaptability and accuracy, which necessitates targeted adjustments. The transformation of data, such as focusing on percentage returns, proved crucial for achieving meaningful predictions. Iterative experimentation with different hyperparameters and tuning algorithms, although challenging, shed light on potential improvements and model behaviors. Moreover, we recognized that computational resources are a significant factor in optimizing models, as they can limit the scope and scale of experiments.

Looking forward, adopting more sophisticated hyperparameter tuning methods like Hyperband, which employs a multi-armed bandit approach, as opposed to HyperOpts tree-structured parzen estimator approach, could enhance efficiency and model performance. Hyperband also natively implements dynamic early stopping and pruning allowing us to not have to manually create these functions. Moving model training to cloud-based platforms, such as Amazon EC2 C5 Instances, would address computational limitations, facilitating larger-scale and more refined optimizations. Additionally, investigating advanced dropout techniques, including Zoneout, Recurrent, and Variational, could further improve the regularization and robustness of our LSTM models, potentially leading to better generalization and lower variance in predictions. This work underscores the potential for machine learning models to inform better trading strategies, mitigate risks, and support data-driven decision-making in volatile financial markets.

References

Zoneout:
[1606.01305] Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations
Recurrent Dropout:
[1512.05287] A Theoretically Grounded Application of Dropout in Recurrent Neural Networks
Variational Dropout:
[1506.02142] Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning

Project Links

Bitcoin-Prediction-Code