Predicting Digital Asset Trends

Aishwarya Iyer, Gay Shinlee, Ryan Lee

March 2025

1 Problem Statement

The primary objective of this project is to forecast the future closing prices of digital assets to construct an optimal cryptocurrency portfolio. Specifically, we aim to develop a predictive model that estimates its closing price for the following day, given a historical sequence of a cryptocurrency's data. Accurate forecasting of asset prices and market dynamics enables dynamic portfolio allocation strategies that seek to maximize returns while minimizing associated risks.

To enhance prediction performance, the model integrates asset-specific features (such as open, high, low, and close prices, trading volume, and market capitalization) with macroeconomic indicators (including the S&P 500 index and treasury yield spread) and sentiment metrics (such as the Fear and Greed Index), enabling more informed investment decisions.

Code Repository

Our codebase can be found in our GitHub repository here. Refer to the `README.md` to navigate the code base.

Our app GUI can be launched from the project root folder by running `streamlit run app.py`. Based on the predictions generated by our model, our app constructs a diversified cryptocurrency basket aimed at maximising potential returns while managing risks. For each cryptocurrency, the model provides a predicted next-day closing price. We calculate the percentage increase (or decrease) between the current closing price and the predicted price. This metric serves as a proxy for the expected short-term performance of each asset. Cryptocurrencies with the highest predicted percentage increases are prioritized and expected to outperform the others in the near term. The basket is diversified across multiple cryptocurrencies to mitigate risk rather than concentrating investments in a single asset. The allocation is weighted proportionally to each asset's predicted percentage increase, ensuring that assets with higher potential receive a larger portfolio share.

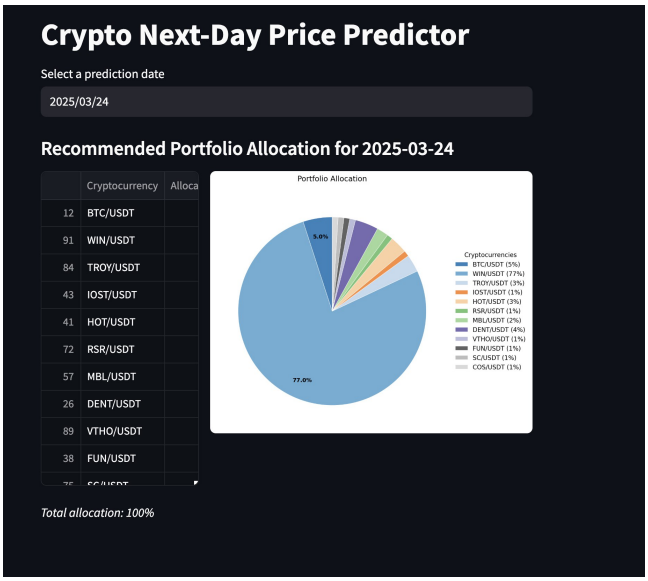

Figure 1: Cryptocurrency Basket App GUI

2 Dataset and Collection

A diverse set of datasets encompassing asset-specific and macroeconomic factors was collected to support robust cryptocurrency price prediction. These datasets capture multiple dimensions of the financial ecosystem influencing digital asset prices:

- **OHLC Data (Open, High, Low, Close):** Daily price movements of various digital assets serve as the primary forecasting input. These features encapsulate intraday volatility and market sentiment, providing rich temporal information necessary for sequence modeling.
- **Volume and Market Cap:** Trading volume reflects market activity and investor interest, while market cap measures an asset's relative size and stability. Together, they provide essential liquidity signals, as highly liquid assets are generally less volatile and more predictable.
- **Daily Returns:** Calculated as the percentage change in closing prices, daily returns provide direct insights into asset volatility and momentum patterns. Modeling returns can often enhance predictive performance compared to modeling raw prices.
- **S&P 500 Index Data:** As a proxy for overall market sentiment and risk appetite, the S&P 500 index can influence investment behavior in alternative assets such as cryptocurrencies. Including this data allows

the model to account for broader financial market dynamics that may indirectly affect digital asset prices.

- **Treasury Yield Spread:** The spread between long- and short-term US Treasury yields is a widely recognized indicator of economic outlook and investor sentiment. A narrowing or negative spread often signals increased market uncertainty, which can drive shifts in demand between traditional and alternative assets.
- **Fear and Greed Index:** Sentiment indicators, particularly those designed for cryptocurrency markets, are critical for modeling the psychological factors influencing asset prices. The Fear and Greed Index quantifies market emotion, ranging from extreme fear (potential buying opportunities) to extreme greed (potential market corrections).
- **Gold Price in USD:** Gold is traditionally viewed as a safe-haven asset. Comparing cryptocurrency movements with gold prices provides a relative measure of risk-off versus risk-on behavior among investors, offering additional context for predicting cryptocurrency trends.

The datasets were obtained from publicly available sources including Yahoo Finance, FRED (Federal Reserve Economic Data), and alternative cryptocurrency data aggregators. By integrating both micro-level (asset-specific) and macro-level (economic and sentiment) variables, the dataset construction aims to provide a comprehensive foundation for modeling the complex, multifactorial drivers of cryptocurrency price movements.

A detailed code walkthrough for the data collection process is provided in `notebooks/data_collection.ipynb`. This script programmatically retrieves data from the sources above, enabling complete reconstruction of the raw datasets from scratch if needed. However, running this notebook is unnecessary, as the preprocessed datasets have already been included in the GitHub repository for convenience.

3 Data Preprocessing

A code walkthrough for data preprocessing can be found in `notebooks/data_processing.ipynb`. The code compiles the raw data and compiles it before creating the train, validation, test sets. Sequence construction is performed internally in the constructor of the `CryptoDataset` object found in `src/dataset.py`.

3.1 Feature Selection

We use 11 input features for each time step: open, high, low, and close prices; trading volume; market cap; daily return; S&P 500 index value; treasury yield spread (10-year minus 2-year); gold price; and the Fear & Greed index. Our prediction target is the closing price of the next time step. The current day's closing price is also included as one of the input features, as this information

would have been fully observable and available to a real-world model at the time of prediction. Only the subsequent day's closing price, which serves as the forecasting target, remains unknown during inference.

We do not use the actual dates as an input feature. This is because temporal relationships between time steps are already captured implicitly through the construction of sequential input windows. Thus, the model can learn temporal patterns without requiring an explicit time encoding.

Initially, we explored incorporating the cryptocurrency ticker (i.e., asset identifier) as an additional feature. A one-hot encoding approach was quickly ruled out, as the dimensionality would have become excessively large given that we collected data for 100 cryptocurrencies. We then attempted a learnable embedding approach, where the model would learn a compact embedding vector for each cryptocurrency during training. While this method avoided the dimensionality explosion, early experiments showed negligible performance gains. Moreover, ticker-based embeddings would not generalize well to cryptocurrencies not seen during training, as unseen assets would have no meaningful learned vector and would need to be masked or handled separately.

Given these observations, and to maintain a compact and generalizable input space, we ultimately chose to omit the ticker information altogether, making the model 'blind' to the specific cryptocurrency type. This design forces the model to rely solely on price and feature dynamics, promoting better generalization across different assets.

3.2 Dataset Time Frame Selection

Although some features, particularly macroeconomic indicators such as gold prices and treasury yields, offer historical records extending as far back as 1978, it was essential to establish a fixed and consistent time frame for constructing our compiled dataset. While long historical series can offer breadth, they may not accurately reflect the structural characteristics and behaviors of the cryptocurrency market, which has only gained significant maturity in recent years. Furthermore, certain digital assets and sentiment indicators have only become available in the last few years, making it necessary to limit the historical range to ensure feature consistency across all assets.

Given the relatively recent emergence and rapid evolution of cryptocurrency markets, the availability of high-quality, stable historical data remains comparatively limited when contrasted with traditional financial assets. To balance the trade-off between dataset size and relevance to present-day market conditions, we selected a three-year window of historical data, spanning from March 24, 2022, to March 24, 2025. This time frame captures a diverse range of market dynamics, including periods of extreme volatility, market corrections, and phases of recovery and growth.

By focusing on a recent three-year period, we ensure that newer digital assets—which have gained significance and trading volume only in recent years—are included. Importantly, applying a consistent window across all cryptocurrencies avoids biases that could otherwise arise from heterogeneous data availability.

Additionally, restricting the historical window to recent data reduces the risk of training the model on outdated market regimes, thereby enhancing the model's ability to learn patterns that are representative of the current structural and behavioral dynamics of cryptocurrency markets. This standardization facilitates coherent model training and evaluation, ultimately supporting more reliable predictive performance.

3.3 Handling Missing Values

In processing the raw datasets, we encountered missing values across several features. These missing entries were not due to errors in the data collection process, but were inherent to the sources' nature. In particular, missing values were observed in the S&P 500 Index and the Treasury Constant Maturity Spread, which do not update on weekends or U.S. public holidays when traditional financial markets are closed. Similarly, the Fear and Greed Index, a sentiment-based metric, occasionally had missing entries due to reporting delays or gaps in publication.

We adopted a **forward-fill** strategy to address these missing values, where the most recently available value is propagated forward until a new observation becomes available. Forward-filling is an appropriate choice for these features because macroeconomic indicators and market sentiment tend to evolve gradually, rather than undergoing abrupt day-to-day changes, particularly over short periods such as weekends. Furthermore, during weekends and market holidays, even market participants and institutional traders operate under the assumption that the most recently published information remains valid until new data is released. In practice, decision-making during these periods inherently relies on forward-propagated values. Thus, forward-filling mirrors the real-world behavior of how financial actors process information gaps, making it not only a technically sound choice but also an economically realistic one. This method preserves the temporal continuity of the data without introducing artificial volatility or noise that could mislead the learning algorithm. Moreover, forward-filling avoids introducing unrealistic assumptions about market behavior during missing periods, making it a conservative yet practical imputation strategy.

A slightly different challenge arose with the gold price data. Although extensive historical gold prices are available, the specific dataset we used only reported values on a monthly basis, typically at the end of each month. To integrate gold prices into our daily-resolution dataset, we forward-filled the most recent available price across subsequent days until the next reported value. This approach is justified because gold, as a traditionally stable asset, tends to exhibit much slower price dynamics compared to the high volatility typical of cryptocurrencies. As a result, forward-filling between monthly intervals provides a reasonable and consistent approximation for modeling purposes.

Overall, the forward-fill strategy was employed to ensure feature continuity, preserve realistic temporal dynamics, and maintain the integrity of the dataset for reliable model training.

3.4 Dataset Partitioning

After performing preliminary data cleaning and merging the datasets (sorting first by ticker, followed by chronological sorting by date), we proceeded to partition the data into three distinct sets: training, validation, and test. This partitioning strategy is vital for developing a predictive model that not only performs well on known data but also generalizes effectively to unseen, future data—mirroring real-world scenarios.

The combined dataset was split using chronological cutoffs, ensuring that the training, validation, and test sets are derived from distinct, non-overlapping time periods. This method is crucial for preventing temporal leakage, where future data inadvertently influences the model's training. Temporal leakage leads to overly optimistic performance estimates and undermines the model's ability to make accurate predictions in the future. By enforcing clear temporal boundaries between the training and test datasets, we ensure that the model is always trained on historical data and evaluated on future, unseen data—similar to real-world deployment, where future prices are inherently unknown.

The chronological split also respects the time-dependent nature of financial data, which is a defining characteristic of markets like cryptocurrency. Unlike random splits, which may inadvertently mix past and future data, a chronological partition maintains the integrity of the data's sequential structure, where market dynamics and price movements evolve over time. If the data were randomly split, the model could 'leak" future information during training, distorting its ability to forecast future events accurately. By partitioning the data before applying the sliding window approach to generate sequences, we ensure that no time step in the training set appears in the test set. This approach mitigates the risk of information leakage and is crucial for maintaining the realism of the evaluation process in time series forecasting tasks.

Furthermore, the chronological split aligns with the concept of backtesting, a widely-used practice in financial modeling where a model is trained on historical data and evaluated on future data that was unavailable during the training period. Backtesting provides valuable insights into how a model would perform in a live trading environment. By adopting this methodology, we enhance the credibility of our evaluation, ensuring that the model's performance is assessed in a manner consistent with real-world financial forecasting.

The dataset was partitioned according to the following time-based splits:

- **Training Set:** From March 24, 2022, to May 31, 2024 (approximately 2 years and 2 months), used to train the model on historical data, allowing it to learn patterns and trends.
- **Validation Set:** From June 1, 2024, to November 30, 2024 (approximately 6 months), used for hyperparameter tuning and model adjustments, ensuring that the model generalizes well to unseen data.
- **Test Set:** From December 1, 2024, to March 24, 2025 (approximately 4 months), used to evaluate the model's final performance and simulate its predictive capabilities on future data.

By organizing the data in this manner, we ensure that the model is not trained on data it will later be evaluated on, thereby providing a more realistic and reliable measure of performance. This approach minimizes the risk of overfitting, as it ensures that the model's predictive power is derived from its ability to capture meaningful trends in the data rather than memorizing specific time periods. Moreover, the use of chronological splits supports the evaluation of the model's ability to adapt to evolving market conditions, an essential characteristic when dealing with financial time series data.

3.5 Sequence Construction

Our combined dataset consists of daily cryptocurrency data, meaning that there is a one-day interval between consecutive time steps. To transform this time series into a supervised learning format, we applied a sliding window approach, segmenting the data into overlapping, fixed-length sample sequences. Each window contains a predefined number of consecutive time steps, and a configurable stride controls the degree of overlap between adjacent sequences.

For supervised learning, the model requires both an input and a corresponding target. Thus, the window size is set to one more than the desired input sequence length: the first part of the window serves as the model input, and the closing price of the final time step serves as the target value. In our case, we selected a window size of 15, corresponding to an input sequence length of 14 days (approximately two weeks), and a stride of 1 day. This setup ensures that each training example is composed of 14 consecutive days of historical data, used to predict the closing price on the following day.

Before constructing sequences, we first partitioned the data by cryptocurrency ticker. Sequences are extracted independently from each cryptocurrency's data to ensure that all time steps within a sequence belong to the same asset. This avoids any unintended mixing of information across different cryptocurrencies, which would be unrealistic for prediction and could introduce misleading patterns during training.

The choice of a relatively short sequence length (14 days) balances multiple considerations. Shorter sequences allow the generation of more training samples and reduce computational burden during model training. Additionally, since our forecasting task is to predict the next day's closing price rather than to model longer-term trends, focusing on recent short-term dynamics is likely more beneficial. Providing excessively long historical sequences could potentially introduce noise or encourage the model to overfit on irrelevant long-term fluctuations, especially given the high volatility and regime shifts characteristic of cryptocurrency markets.

Overall, this sequence construction approach transforms the raw daily data into a structured supervised learning dataset while preserving the temporal dependencies crucial for effective time series forecasting.

3.6 Feature Normalization

To enhance model training stability, we applied feature normalization to the input data. Specifically, we standardized each feature by subtracting its mean and dividing by its standard deviation, so that each feature has approximately zero mean and unit variance.

Normalization is particularly important for time series forecasting models because it ensures that features with larger numerical ranges do not dominate the learning process. Without normalization, features on vastly different scales could lead to unstable or inefficient gradient updates, slowing down optimization and potentially resulting in suboptimal model performance. Standardization also helps the model learn smoother, more balanced representations across all input variables, making training more stable and effective.

Crucially, the normalization parameters—the mean and standard deviation for each feature—were computed solely using the training set. These parameters were then used to normalize the validation and test sets, and should be used to normalize any input data during inference. Normalizing with respect to only the training set prevents data leakage from the validation and test sets into the training process, ensuring that model evaluation remains unbiased and realistic. Using the training set's mean and standard deviation to normalize is important as the model would be trained on the training set's distribution.

4 Model Architecture

We experimented with four different deep learning architectures for the cryptocurrency price prediction regression task. All models were implemented in PyTorch and can be found in `src/models.py`.

4.1 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are a natural choice for modeling time series and sequential data. Unlike feedforward networks, RNNs maintain a hidden state that is updated at each time step, allowing them to capture temporal dependencies and patterns across sequences. This property makes them particularly well-suited for financial forecasting tasks such as cryptocurrency price prediction, where the evolution of market dynamics over time is crucial for accurate modeling.

However, traditional vanilla RNNs are known to struggle with learning long-term dependencies due to challenges such as the vanishing and exploding gradient problems. Gradients can either decay to near-zero values or grow uncontrollably during backpropagation through long sequences, making effective training difficult. Traditional residual connections across time steps are not directly applicable to recurrent models due to the strict temporal dependencies between time steps (though they can still be used between stacked RNN layers). To mitigate these issues, more advanced recurrent architectures like Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks

were developed. These architectures introduce gating mechanisms that regulate the flow of information across time steps, enabling better retention of long-term dependencies.

For our RNN-based models, after processing the entire input sequence, we extract the hidden state at the final time step and pass it through a fully connected linear layer to produce the regression output (i.e., the predicted closing price for the next day).

Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks are a widely-used extension of standard RNNs that incorporate memory cells and gating mechanisms—specifically, input, output, and forget gates. These gates allow the network to selectively retain or discard information across time steps, addressing the vanishing gradient problem and enabling the modeling of longer-range temporal dependencies.

In our experiments, we implemented an LSTM model where the input sequence is passed through one or more stacked LSTM layers. We included a hyperparameter toggle to control whether the LSTM operates in a unidirectional or bidirectional mode. In bidirectional LSTMs, two separate LSTM layers are trained: one processes the sequence in the forward direction (past to future) and the other processes it in the reverse direction (future to past). The hidden states produced by the forward and backward passes at each time step are concatenated before being passed to the final prediction layer. Bidirectional models allow the network to capture both past and future context within the input window, which can be particularly advantageous when the target depends not just on past information but also on broader sequence-level patterns.

Gated Recurrent Unit (GRU) Networks

Gated Recurrent Unit (GRU) networks are a simplified variant of LSTMs that combine the forget and input gates into a single update gate and merge the cell state and hidden state. This results in a more compact architecture with fewer parameters, potentially making GRUs faster and easier to train, especially on smaller datasets or when computational efficiency is important.

Like the LSTM model, our GRU implementation also supports both unidirectional and bidirectional processing. The final hidden state from the last time step (or concatenated final states in the case of a bidirectional GRU) is passed through a fully connected layer to generate the regression prediction. GRUs are often found to perform comparably to LSTMs for shorter sequence modeling tasks while being computationally lighter. This makes it a promising architecture for our experiments, in which we provide a relatively short input sequence (14 time steps).

4.2 Transformers

While recurrent neural networks (RNNs) process sequential data step-by-step, Transformers operate entirely through self-attention mechanisms, enabling them to model dependencies between any two time steps in a sequence regardless of their temporal distance. By removing the inherent sequential bottleneck of RNNs, Transformers allow for efficient parallelization during training and can more effectively capture complex, long-range temporal relationships. Owing to their success across a broad range of sequence modeling tasks, including natural language processing and time series forecasting, we explored Transformer-based architectures for cryptocurrency price prediction.

In contrast to RNNs, which compress sequence history into a hidden state, Transformers use self-attention to compute pairwise interactions between all input tokens simultaneously. For each input element (or time step), the model generates three vectors: a query Q, a key K, and a value V. The attention (the scaled dot product variant) scores are computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where d_k is the dimensionality of the key vectors. This mechanism allows each position to attend to all other positions, enabling the model to contextualize each time step based on the entire input sequence with direct connections.

Although Transformers are often associated with modeling long-range dependencies, in our experiments the input sequence length was relatively short (14 days). Nonetheless, we hypothesized that the Transformer architecture could still offer significant advantages. Specifically, the explicit modeling of pairwise relationships via self-attention allows the network to dynamically prioritize relevant portions of the input history without being constrained to sequential processing. Even within short sequences, cryptocurrency markets may exhibit complex interactions — such as rapid trend reversals, local volatility patterns, or momentum shifts — which may not be easily captured by simple sequential models. Moreover, by decoupling representation learning from strict temporal adjacency, Transformers can better capture global structure and higher-order interactions among input features. For example, the model can learn to simultaneously attend to both very recent prices and slightly older key events within the two-week window, enabling richer and more granular contextualization for prediction purposes. While the relative benefits of Transformers over simpler architectures (like GRUs or LSTMs) may be less dramatic for shorter input lengths, we hypothesize that the expressiveness of self-attention with its capabilities for global context aggregation, can nonetheless improve model robustness and generalization.

Base Transformer Encoder

Our cryptocurrency price prediction task is formulated as a many-to-one sequence regression problem: given a sequence of historical observations, the

model predicts the closing price at the next time step. Since we are interested only in predicting a single future value rather than generating an output sequence, we utilize only the encoder component of the Transformer architecture.

Our base Transformer model closely follows the original architecture proposed by Vaswani et al. [2]. The model begins by projecting the raw input features — consisting of 11 financial indicators — into a higher-dimensional embedding space via a linear transformation. As the self-attention mechanism is inherently permutation-invariant, we inject positional information into the embeddings using fixed sinusoidal positional encodings. This allows the model to maintain awareness of the relative and absolute positions of each time step within the sequence.

The embedded inputs, augmented with positional encodings, are then passed through a stack of Transformer encoder layers. Each encoder layer consists of a multi-head self-attention mechanism (scaled dot product attention), a position-wise feedforward network, residual (skip) connections, and layer normalization applied after each sublayer. This layered structure enables the model to dynamically attend to important temporal patterns across the sequence while maintaining stable training dynamics. After processing the sequence through the encoder stack, the output embeddings — now enriched with global contextual information — are utilized for the final prediction.

Informer Model

We also created a variant model based on the Informer model proposed by Zhou et al. [3], which introduces several key innovations tailored for time-series forecasting.

One major innovation is the use of *ProbSparse* self-attention, a sparse attention mechanism that reduces the computational complexity of full attention (originally quadratic). Instead of computing attention across all query-key pairs, the Informer selectively attends to a subset of "dominant" queries based on the sparsity assumption that only a few time steps are truly informative. Specifically, for each query q_i, attention scores are computed as:

$$\text{Attention}(q_i, K, V) = \sum_{j \in \mathcal{S}(q_i)} \text{softmax}\left(\frac{q_i^\top k_j}{\sqrt{d_k}}\right) v_j,$$

where $\mathcal{S}(q_i)$ denotes the subset of keys k_j selected based on the magnitude of $q_i^\top k_j$. By focusing only on the most relevant keys for each query, the Informer reduces attention computation to $\mathcal{O}(L \log L)$ instead of $\mathcal{O}(L^2)$, where L is the sequence length.

The second innovation is *convolutional distillation*, applied between successive encoder layers. Instead of passing the full output sequence between layers, one-dimensional convolutional operations are used to progressively downsample the sequence length while preserving essential temporal information. This

mechanism acts as a form of local feature extraction and aggregation, encouraging the model to compress and abstract useful patterns over adjacent time steps. Convolutional layers are particularly well-suited for capturing local dependencies and invariant patterns, which can be beneficial in time series data where nearby points are often highly correlated.

Although our input sequences are relatively short (14 days), these design choices still offer potential advantages. ProbSparse attention encourages the model to focus on particularly informative days, which may be crucial given the noisy and volatile nature of financial time series. Convolutional distillation in our setting serves a purpose beyond simple downsampling. Early empirical observations suggest the presence of strong local correlations between consecutive daily observations in cryptocurrency price data. Convolutional layers, with their inherent inductive bias toward local feature extraction, are particularly well-suited to exploit such short-range dependencies. By applying convolutional distillation between encoder layers, the model can robustly and efficiently extract localized temporal patterns, improving noise tolerance and enhancing feature abstraction in the early stages of the network. Meanwhile, the self-attention mechanism operates at a complementary scale, modeling broader, global relationships across the entire input sequence. This interplay between localized convolutional aggregation and global attention-based modeling enables a richer and more hierarchically structured representation of the temporal dynamics present in the data.

For our implementation of convolutional distillation, instead of using a two-layer position-wise feedforward network in each encoder layer, we replace it with two sequential 1-D convolutional layers. The first convolution expands the feature dimension, applies a non-linear activation (ReLU or GELU), and the second projects it back to the original size. Both convolutional layers use a kernel size of 1, effectively acting as learnable feature-wise transformations while enabling local feature aggregation.

Learnable Special Token

Inspired by the BERT architecture [1], our models prepend a special `[CLS]` token to the beginning of every input sequence before processing. This token is implemented as a learnable embedding vector, which is concatenated with the projected feature embeddings before positional encoding is applied. After the sequence (now including the special token) is processed by the encoder, we extract the final contextualized representation of the `[CLS]` token and pass it through a linear layer to produce the final regression prediction.

The `[CLS]` token participates in the self-attention mechanism like any other token, attending to and being attended by all other time steps in the sequence. The model essentially learns a good initial embedding vector for this special token through training, before it attends to other tokens in the sequence. The intuition behind this design is that the model learns a good representation of `[CLS]` such that it acts as a dedicated aggregate representation of the entire sequence. During training, the network learns how to encode global, task-relevant

information into the [CLS], allowing it to dynamically aggregate information across the entire input through attention-based interactions. Crucially, the learnable initial embedding of the `[CLS]` token enables the network to adaptively shape how this aggregation occurs, learning a dedicated sequence-level summary representation that is optimized for the prediction task. By relying on attention rather than manual pooling (e.g., mean or max pooling), the model is able to weigh different parts of the sequence according to their task relevance, potentially leading to more expressive and task-specific representations.

We follow the BERT convention of always prepending the `[CLS]` token to the start of every sequence, ensuring that its position remains fixed across all inputs. This consistency helps stabilize its positional encoding throughout training. Since the encoder uses fully bidirectional attention (i.e., no causal masking), the `[CLS]` token can freely incorporate information from the entire sequence without problem. While the `[CLS]` token was conceived for classification tasks, we found that it works well for regression tasks as well.

5 Evaluation Methodology

5.1 Training Details

A code walkthrough of our training process can be found in `notebooks/model_training.ipynb`. As discussed in earlier sections, we partitioned the dataset chronologically into training, validation, and test sets.

- **Training Set:** From March 24, 2022, to May 31, 2024 (approximately 2 years and 2 months)
- **Validation Set:** From June 1, 2024, to November 30, 2024 (approximately 6 months)
- **Test Set:** From December 1, 2024, to March 24, 2025 (approximately 4 months)

This temporal split was critical to maintaining the integrity of our forecasting task, ensuring that the models are always evaluated on future, unseen data. In particular, the test set consists of the most recent time periods, providing a realistic assessment of the models' ability to generalize to truly unseen future market conditions.

For sequence extraction, we applied a sliding window approach separately to each set, using a window size of 15. Each window consists of the preceding 14 days of features as input, and the closing price on the 15th day as the target. A stride of 1 was used to maximize the number of available training samples. While hyperparameters such as sequence length and stride could be further tuned, we fixed these values for all experiments to maintain consistency. The choice of 14 days balances the trade-off between providing enough historical context for meaningful prediction and maintaining a sufficient number of training examples,

while avoiding excessive sequence lengths that may introduce unnecessary noise or overfitting risks.

We used a batch size of 256 throughout all experiments. This batch size was chosen as it provided a practical balance: large enough to stabilize gradient estimates and benefit from GPU acceleration, yet small enough to avoid memory overflow and to ensure reasonable wall-clock training times given our computational resources.

Due to limited time and computational constraints, we conducted only basic hyperparameter tuning. The Adam optimizer, with a learning rate of 0.001, consistently yielded the best results across preliminary experiments. Although we briefly tested other PyTorch optimizers such as SGD, RMSProp, and AdamW, we observed only minor differences in performance. Importantly, larger learning rates often led to unstable training dynamics, characterized by oscillatory behavior and frequent spikes in both training and validation losses. Since the training loss increased as well, this instability was not due to conventional overfitting, but rather to overly aggressive parameter updates causing the optimizer to overshoot local minima. While occasional escapes from sharp local minima can be beneficial for generalization, the magnitude of instability observed at higher learning rates was detrimental to convergence stability.

L2 regularization was applied via the `weight_decay` parameter of the Adam optimizer, with 1e-5 found to be a suitable value after empirical testing. We also experimented with L1 regularization by manually incorporating an $L1$ penalty term into the loss function. However, given the marginal differences observed and the simplicity of implementation, we ultimately adopted L2 regularization across all experiments for the purposes of mitigating overfitting.

We used PyTorch's `torch.nn.MSELoss`, corresponding to Mean Squared Error (MSE), for the loss function (training objective). MSE is a natural and standard choice for regression tasks, as it penalizes larger errors more heavily, promoting models that consistently predict close to the ground truth values. Moreover, MSE's smooth, differentiable nature facilitates stable optimization via gradient-based methods.

5.2 Evaluation Metrics

To evaluate model performance, we tracked three main metrics: Mean Absolute Error (MAE), R^2 score (coefficient of determination), and explained variance.

Mean Absolute Error provides a direct measure of average prediction error in the same units as the target variable. Although MAE is useful for monitoring relative changes across epochs (e.g., detecting overfitting or underfitting trends), it is less interpretable in isolation for assessing model quality, especially when the target variable has non-trivial variance over time.

The R^2 score was selected as our primary evaluation metric. The R^2 score quantifies the proportion of variance in the target variable that is explained by the model's predictions, relative to a simple baseline (predicting the mean). It naturally accounts for both the scale and variability of the data, making it highly interpretable for regression settings. A score of $R^2 = 1$ indicates perfect

prediction, whereas a score of $R^2 = 0$ implies that the model performs no better than predicting the mean value.

Finally, explained variance provides a complementary view, measuring the proportion of variance captured by the model without penalizing systematic bias as strongly as R^2. While both metrics are closely related, tracking explained variance can sometimes highlight cases where the model captures the overall structure well but suffers from slight bias.

By tracking all three metrics, we obtain a comprehensive understanding of both prediction accuracy and the underlying quality of the learned representations.

5.3 Best Model Hyperparameters

Through limited but targeted hyperparameter tuning, we identified the following configurations as yielding the best validation performance for each model:

- `CryptoGRU`: `hidden_size=64`, `num_layers=3`, `dropout_prob=0.1`, `bidirectional=True`
- `CryptoLSTM`: `hidden_size=64`, `num_layers=3`, `dropout_prob=0.1`, `bidirectional=True`
- `CryptoTransformer`: `hidden_size=64`, `num_layers=4`, `dropout_prob=0.1`, `num_heads=4`, `dim_feedforward=256`
- `CryptoInformer`: `hidden_size=64`, `num_layers=4`, `dropout_prob=0.1`, `num_heads=4`, `dim_feedforward=256`, `probsparse_sampling_factor=5`

For the recurrent models (GRU and LSTM), we observed that the bidirectional architectures consistently outperformed their unidirectional counterparts. This is likely because bidirectional processing enables the model to capture dependencies in both forward and backward temporal directions, which is particularly beneficial when predicting financial time series where contextual information from both past and (near) future observations can help refine feature representations. Furthermore, increasing the number of recurrent layers beyond three did not yield meaningful performance improvements; instead, it often introduced unnecessary complexity and slower convergence.

A similar trend was observed for the Transformer-based models, where increasing the number of encoder layers beyond four showed diminishing returns. In particular, deeper Transformer models tended to require more aggressive regularization and longer training times, without corresponding gains in validation performance.

Due to the constraints of limited compute and available time, we did not conduct an exhaustive hyperparameter search across all possible dimensions, such as larger hidden sizes, or alternate attention head configurations. We leave a more comprehensive exploration of these hyperparameters to future work, which may uncover further performance gains.

It is important to emphasize that all hyperparameter choices were made exclusively based on validation set performance. The test set was kept strictly

isolated until the final evaluation phase, following best practices to avoid data leakage. Premature exposure to the test set during model development can lead to overly optimistic performance estimates, as it introduces information from unseen future data into model selection decisions. By holding out the test set entirely during tuning, we ensure that final performance metrics provide an honest assessment of each model's true generalization capabilities.

6 Results and Discussion

After identifying the optimal hyperparameters through our tuning process, we retrained each model architecture using these best-found settings. For the Transformer-based models, we trained for 25 epochs. Although validation performance generally plateaued around the 15-epoch mark, we extended training to 25 epochs to allow for the possibility of late-stage improvements and to account for potential fluctuations in performance due to training dynamics.

For the recurrent architectures (GRU and LSTM variants), we trained the models for 50 epochs. In contrast to the Transformer models, the recurrent models' validation performance continued to improve, albeit slowly, beyond the 15-epoch range and had not fully plateaued by epoch 50. Based on the validation loss curves, we believe that further training could potentially yield additional performance gains. However, given the significantly longer training times required for recurrent networks and the relatively lower performance observed compared to Transformer models, we made the practical decision to stop training at 50 epochs to balance time, computational cost, and diminishing returns. The results for each model are shown in Table 1.

Model Architecture	**MAE** $\downarrow$	$\mathbf{R^2}$ $\uparrow$	**Explained Variance** $\uparrow$
Bidirectional GRU	1007.390	0.0266	0.0379
Bidirectional LSTM	1036.893	0.0221	0.0331
Transformer	377.856	0.901	0.902
Informer	351.780	**0.911**	0.913

Table 1: Model performance on the test set. Lower MAE and higher R^2/Explained Variance indicate better performance.

Our experimental results demonstrate that Transformer-based models consistently outperform recurrent models across all evaluation metrics, with the Informer-based model achieving the best overall performance. These findings align with our earlier hypotheses regarding the advantages of Transformer architectures, even when applied to relatively short input sequences (14 time steps). Notably, the superiority of Transformer-based models cannot simply be attributed to an increase in parameter count. We observe that increasing the depth (i.e., number of layers) in recurrent architectures such as GRUs and LSTMs did not lead to significant performance gains, suggesting that the inherent architectural properties of Transformers—particularly their ability to model global dependencies through self-attention—play a more critical role. The GRU seems to perform better than the LSTM, likely because it is better at modeling short-term dependencies.

This supports the view that attention mechanisms offer substantial benefits over recurrent inductive biases, even for short-term sequential data. The recurrent models appear limited by their sequential processing nature and reliance on hidden state propagation, whereas Transformers, with their global receptive field, more effectively capture complex interactions between different time steps without being constrained by sequential bottlenecks.

Future work could extend this study in several directions. First, we could explore more recent Transformer variants designed for enhanced efficiency and performance, such as Longformer, Performer, or FlashAttention-based models, particularly when scaling up sequence lengths. In this work, we kept the sequence length fixed (15 days) for simplicity and comparability across models; however, increasing the input sequence length could further stress-test the models' ability to capture long-range dependencies. Evaluating how well Transformer architectures maintain their performance with significantly longer input sequences would provide valuable insights.

Moreover, for handling longer sequences, it would be worthwhile to investigate the use of convolutional tokenizers—lightweight 1D convolutional layers that compress input sequences before feeding them into Transformer encoders. This approach could reduce the computational burden associated with longer sequences while preserving important local structures in the data.

Another promising avenue would be experimenting with hybrid architectures that combine recurrent layers with attention mechanisms, such as Transformer-RNN hybrids or attention-augmented RNNs. Additionally, more extensive hyperparameter tuning (e.g., hidden dimensionality, dropout rates, attention head counts) and model regularization strategies (e.g., stochastic depth, label smoothing) could further enhance model performance.

Finally, evaluating model robustness under distribution shifts, such as highly volatile market conditions, would be crucial for understanding generalization in real-world deployment settings. Incorporating uncertainty estimation techniques into the models could also improve their practical applicability to high-risk domains like financial forecasting.

Beyond model architecture improvements, there are also several promising directions for enhancing the dataset itself. One avenue is more thorough feature

engineering; for example, incorporating features such as whether a given day falls on a Friday or a weekend-adjacent trading day, which could potentially influence market behavior (though further analysis would be needed to confirm the significance of such effects). Furthermore, despite our efforts, we were unable to incorporate sentiment indicators derived from social networks into our data set. Practical limitations, including API policy changes, high data access costs, and computational constraints (particularly given the demands of natural language processing), prevented their inclusion. Nevertheless, we believe that sentiment signals extracted from social media could capture real-time market sentiment shifts that are not reflected in traditional sentiment indicators, offering a valuable complementary source of predictive information.

Finally, we note that the task of predicting next-day closing prices might be relatively easier compared to forecasting long-term trends. Short-term predictions primarily rely on recent data, which is more stable and less prone to external influences. In contrast, long-term forecasting introduces significant complexity as it must account for greater market volatility, macroeconomic factors, and unforeseen global events. Future work could involve adapting these models for long-term forecasting, which would require further refinement of the feature set, model architecture, and handling of uncertainty. This direction can potentially tackle more challenging and dynamic aspects of financial prediction.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[3] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting, 2021.