Predicting Future Cryptocurrency Price Lows using Deep Learning

FIE458: Deep Learning and LLMs with Applications to Finance

Spring 2025

Candidate Number: 11, 24, 37

Handed out:	March 22, 2025
Deadline:	June 10, 2025

Final Report

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

The cryptocurrency market, renowned for its dynamic growth and pronounced volatility, presents unique challenges and opportunities for investors. High-capitalization assets like Ethereum (ETH-USD) are particularly susceptible to rapid price fluctuations, underscoring the critical need for robust risk management frameworks and predictive modeling tools. Deep learning (DL) has emerged as a promising frontier in financial forecasting, offering methodologies capable of discerning intricate, non-linear patterns within complex time series data [13]. Recent advancements in the field, exemplified by studies such as Akouaouch and Bouayad (2025) [2], have demonstrated the predictive potential of specialized DL architectures, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, for cryptocurrency price movements. Critically, such research often highlights the necessity for continued exploration into more sophisticated DL architectures and the integration of diverse, correlated features to fully capture latent market interdependencies. This call for expanded investigation forms a core motivation for the present study.

1.2 Problem Statement and Objectives

This research confronts the challenge of forecasting significant downside risk in the Ethereum market, with a specific focus on predicting impending daily price lows. The primary objective is to develop, evaluate, and compare the predictive efficacy of several deep learning architectures. While drawing inspiration from the types of recurrent models explored by Akouaouch and Bouayad (2025) and others, this study expands the investigation to include not only LSTM and GRU networks but also a hybrid LSTM-GRU architecture, and potentially other sequential models. These models are selected for their proven capacity to capture temporal dependencies inherent in financial time series. A further key objective is to assess the impact of an enriched feature set on predictive performance. This involves augmenting the model input with lagged price data from Bitcoin (BTC-USD) as a broader crypto market indicator, alongside macroeconomic data (10-Year U.S. Treasury yield), cryptocurrency market sentiment (Crypto Fear & Greed Index), and traditional market performance (S&P 500 index). The ultimate aim is to determine whether these advanced, feature-augmented DL models can provide reliable and actionable foresight into potential price troughs for Ethereum.

1.3 Contributions

This study contributes to the evolving body of knowledge on DL-based risk forecasting in the cryptocurrency sphere through the following:

- It provides a comparative empirical analysis of several deep learning architectures, specifically LSTM, GRU, and hybrid LSTM-GRU models, applied to the nuanced task of predicting daily price lows for Ethereum, moving beyond general price direction forecasting.
- It builds upon existing research by systematically evaluating the incremental predictive power gained from a comprehensive suite of cross-asset and market sentiment features, in addition to the target asset's own historical data.
- By focusing on the prediction of price troughs, this work offers insights directly applicable to downside risk assessment, stop-loss placement, or strategic entry point identification in a highly volatile market environment.
- The findings aim to provide a more granular understanding of the predictive capabilities and inherent limitations of the investigated DL models in this specific financial forecasting context.

1.4 Report Outline

The structure of this report is as follows:

- **Section 2** outlines the data acquisition pipeline and presents the full scope of the feature engineering process, including the construction of target variables and the derivation and selection of relevant predictors.
- **Section 3** elaborates on the model development lifecycle. This includes a detailed account of the neural network architectures utilized, the final data structuring steps prior to modeling, and the training and validation procedures implemented.
- **Section 4** reports the empirical findings obtained from experimental evaluation, highlighting model performance across relevant metrics and validation regimes.
- **Section 5** provides an in-depth discussion of the results, interpreting their analytical and practical implications. It also considers limitations of the study and outlines potential directions for future research.
- **Section 6** concludes the report with a summary of the key insights and final reflections.

2 Data and Feature Engineering

The foundation of any robust predictive modeling endeavor lies in the meticulous acquisition and thoughtful engineering of data. This section details the multi-stage process undertaken to construct the dataset for this study. It begins with the sourcing and initial preprocessing of primary cryptocurrency data and supplementary external indicators. Subsequently, it elaborates on the critical construction of our target variables—the phenomena we aim to predict—and the predictor variables (features) used by the models. Throughout this process, paramount importance is given to strategies for preventing data leakage, ensuring the integrity and reliability of our model evaluation, a principle heavily emphasized in statistical learning literature (e.g., ISLP, Chapter 2, p.28; Chapter 5, p.201ff) [15].

2.1 Data Acquisition

The initial phase involved assembling a comprehensive dataset by sourcing historical cryptocurrency price data and augmenting it with relevant external economic and market sentiment indicators.

2.1.1 Primary Cryptocurrency Data

Historical daily price data—comprising Open, High, Low, Close, and Volume (OHLCV)—for a range of cryptocurrencies, with a primary focus on Ethereum (ETH-USD), were obtained from the "Top 1000 Cryptos Historical Data" dataset available on Kaggle Hub [17]. Initial preprocessing steps for this raw data included:

1. **Date Parsing:** Converting date strings into a consistent `datetime` format using `pandas.to_datetime()` to facilitate time-based operations.
2. **Column Standardization:** Renaming columns to a uniform convention (e.g., 'Symbol' to `symbol`, 'Date' to `date`) for ease of use.
3. **Chronological Sorting:** Ordering the data chronologically for each cryptocurrency symbol using `df.sort_values(by=['symbol', 'date'])` to ensure correct temporal sequencing for time series analysis.

To focus the analysis on a period with more relevant market dynamics, a lower temporal cutoff was applied to the primary cryptocurrency data. Only data from January 1, 2020, onwards was utilized. The rationale for this cutoff was the observed lower trading activity and less price variation for Ethereum in the period before 2020, which was deemed potentially to introduce less relevant patterns or noise compared to the more recent market behavior on which the models were to be trained. This start date is also reflected in the description of the training set in Section 4.1.

2.1.2 External Economic and Sentiment Data

To provide broader market context and potentially capture influences beyond the target cryptocurrency's own price history, several external time series were incorporated. These external factors were merged with the primary cryptocurrency data based on their respective dates. It is important to note that lagged values of these external series were generated and merged before any asset-specific data splitting to maintain data integrity, as these global indicators are considered exogenous at a given point t for predicting a specific crypto's future. The sourced external data included:

- **Bitcoin (BTC-USD) Closing Prices:** Given Bitcoin's significant influence and high correlation with the broader cryptocurrency market, as shown by multiple empirical studies [22, 9], its closing prices serve as a proxy for overall crypto market sentiment. These were extracted from the main Kaggle dataset.
- **10-Year U.S. Treasury Constant Maturity Rate (DGS10):** This key interest rate, reflecting broader macroeconomic conditions and risk appetite, was obtained from the Federal Reserve Economic Data (FRED) service via its CSV download API[12].
- **S&P 500 Index (`ĜSPC`) Closing Prices:** Sourced using the `yfinance` library, the S&P 500 index provides a measure of general traditional market sentiment, which can influence more speculative asset classes like cryptocurrencies [5].
- **Crypto Fear & Greed Index:** This sentiment indicator, specific to the cryptocurrency market, was fetched from the `alternative.me` API. The index quantifies market sentiment on a scale from Extreme Fear [0-25], Fear [25-45], Neutral [45-55], Greed [55-75], to Extreme Greed [75-100]. It is derived from five key signals: price trends of top cryptocurrencies, volatility forecasts for BTC/ETH, derivatives market activity (puts vs. calls), market balance (Bitcoin's value vs. stablecoins via SSR ratio), and user activity including social media trends [4].

2.2 Target Variable Construction

Defining what the model aims to predict is a crucial step. For this study, we formulated two distinct target variables for each observation at time t:

2.2.1 Definition of Binary and Regression Targets

1. **Binary Target (`binary_target`):** This categorical variable indicates whether the price of Ethereum is expected to experience a "significant" drop within a defined future window (7 days in this study). A drop is deemed significant if the future minimum low price within the window $[t+1, t+W]$ falls below a dynamically determined percentage threshold, T_{pct}, relative to the price at t.
    - Let P_t be the closing price at time t.
    - Let $L_{t,W}$ be the minimum 'Low' price observed in the window from $t+1$ to $t+7$.
    - The percentage change to this future minimum is $C_{t,W} = \frac{L_{t,W} - P_t}{P_t} \times 100$.
    - The binary target is then $Y_{bin,t} = \mathbf{1}(C_{t,W} \leq T_{pct})$, where $\mathbf{1}(\cdot)$ is the indicator function.

    The interpretability is direct: $Y_{bin,t} = 1$ signals a predicted significant drop.

2. **Regression Target (`days_to_future_min_7d`):** This continuous variable represents the number of days from time t until the aforementioned minimum low price $L_{t,W}$ occurs within the 7-day window.

- Let $D_{t,W}$ be the date on which $L_{t,W}$ is observed.
- The regression target is $Y_{reg,t} = (D_{t,W}, \text{date}_t)$, expressed in days.

The calculation of these target variables was carefully performed after the chronological splitting of data into training, validation, and test sets. This ensures that information from the validation or test sets (i.e., future data) does not leak into the target variable calculations for the training set, a critical step for preventing look-ahead bias.

2.2.2 Determining the Percentile Threshold (T_{pct})

The threshold T_{pct} for the binary target is pivotal, as it defines what constitutes a "significant" price drop and directly influences class balance. An overly extreme threshold (e.g., 5th percentile of $C_{t,W}$) could lead to severe class imbalance, with very few instances of $Y_{bin,t} = 1$.

To select an appropriate T_{pct}, we conducted an empirical analysis. The threshold was defined as a specific percentile of the distribution of $C_{t,W}$ values, calculated exclusively on the training dataset to prevent data leakage. We experimented with various percentile values (e.g., 10th to 60th percentile in 10% increments) and evaluated their impact on:

- **Class Imbalance:** The resulting proportion of positive to negative classes for the binary target (visualized in Figure 3).
- **Actual Drop Magnitude:** The monetary percentage drop corresponding to each percentile threshold (illustrated in Figure 4).
- **Distribution of Regression Target:** How the distribution of `days_to_future_min_7d` changes for instances where $Y_{bin,t} = 1$ under different thresholds (shown in Figure 5).
- **Preliminary Model Performance:** The impact on key metrics (e.g., Binary Accuracy, ROC AUC, Regression MAE) using a baseline model (Figure 6).

Initial experiments with the LSTM-GRU model revealed that very low thresholds (e.g., 10-20th percentile) led the model to predominantly predict "No Drop," while intermediate thresholds (e.g., 30th percentile) sometimes caused the model to predict "Drop" too frequently, indicating sensitivity to this parameter and potential challenges with weak signals or insufficient regularization. The MAE for the regression target also showed variability.

Considering these observations and the practical implications, the final selection of T_{pct} was guided by a balance between achieving a reasonable class distribution, defining a financially meaningful "significant drop" (e.g., a drop severe enough to warrant action), and the preliminary model performance.

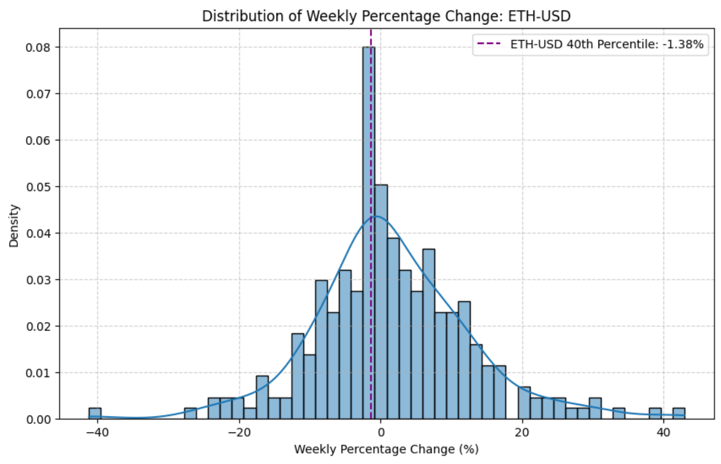

Figure 1: Distribution of Normalized Weekly Price Percentage Change for Ethereum

Table 1: Skewness and Kurtosis of Daily and Weekly Returns for S&P500 and ETH-USD

Asset	**Period**	**Skewness**	**Kurtosis**
S&P500	Daily	-0.051	6.359
	Weekly	-0.109	1.144
ETH-USD	Daily	0.174	4.739
	Weekly	0.308	1.852

The distribution of ETH-USD weekly returns displays a positive skewness (0.308), indicating a propensity for sharper positive returns compared to negative ones. This rightward skew suggests that, while large rallies occur more frequently, the asset remains susceptible to abrupt, severe drawdowns. The differing distributional characteristics justify a divergent approach in defining significant price movements. ETH-USD's positive skewness and fat tails necessitate a higher percentile threshold to preemptively identify emerging downturns.

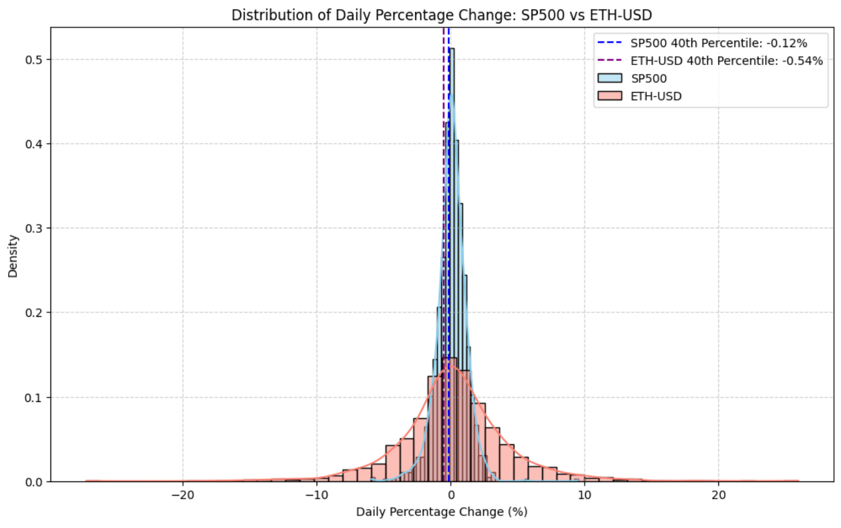

Figure 2: Distribution of Daily Percentage Change in Price for ETH-USD and S&P500, highlighting the fatter tails of ETH-USD

The 40th percentile is justified for ETH-USD due to its positive skew and fat-tailed nature, ensuring that a given trading strategy can use model outputs to properly respond to meaningful downside risk without being overly conservative. For analyzing market movements on an established index like the S&P 500, such a high percentile would be too sensitive, but for crypto, it is a prudent adjustment to account for asymmetric tail risks. This aligns with crypto's high volatility, where waiting for deeper percentiles could mean missing a large part of a crash. This also offered a good trade-off between class balance and signal strength for our ETH-USD training data. The determined value of $T_{pct} = 40$ value from the training set, was then fixed and applied to the validation and test sets.

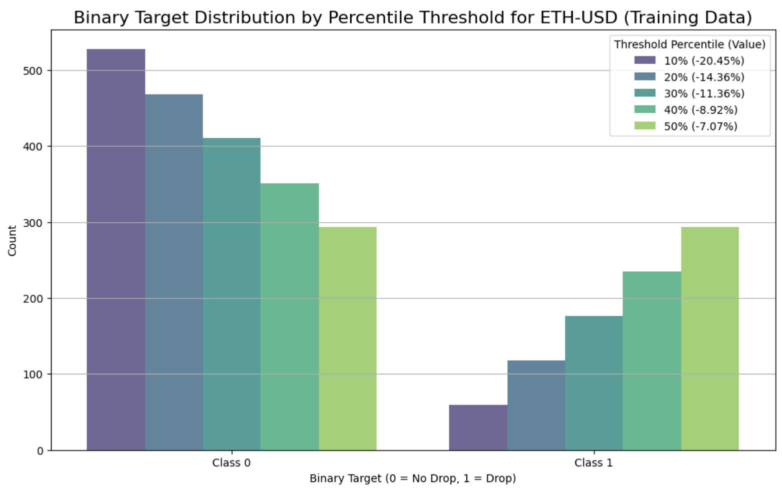

Figure 3: Degree of Class Imbalance for Different Percentile Thresholds of Future Minimum Percentage Change (Training Data).

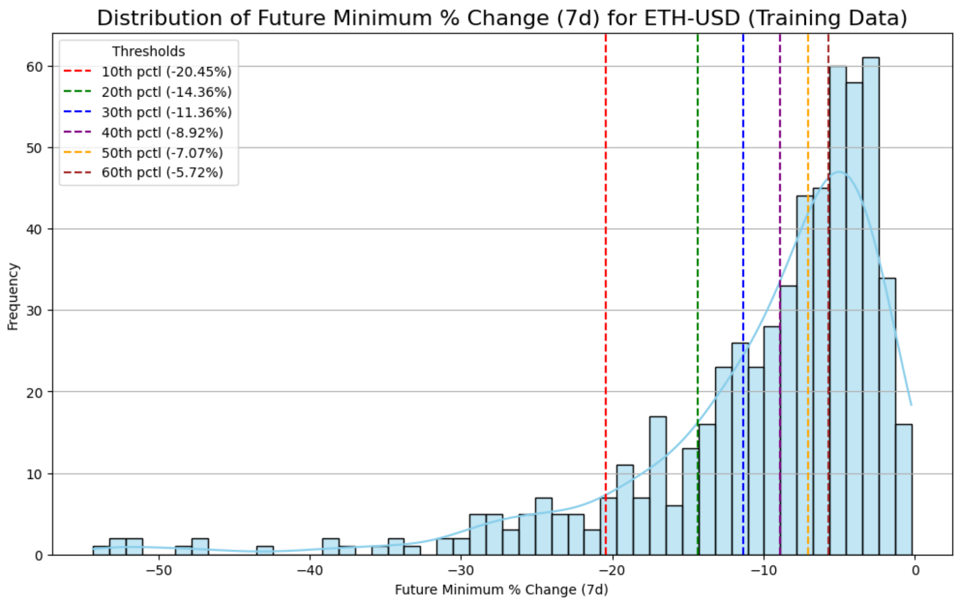

Figure 4: Corresponding monetary percentage drop for different percentile thresholds for $C_{t,W}$ (Training Data). The value in parentheses indicates the actual percentage drop defining the boundary for that threshold.

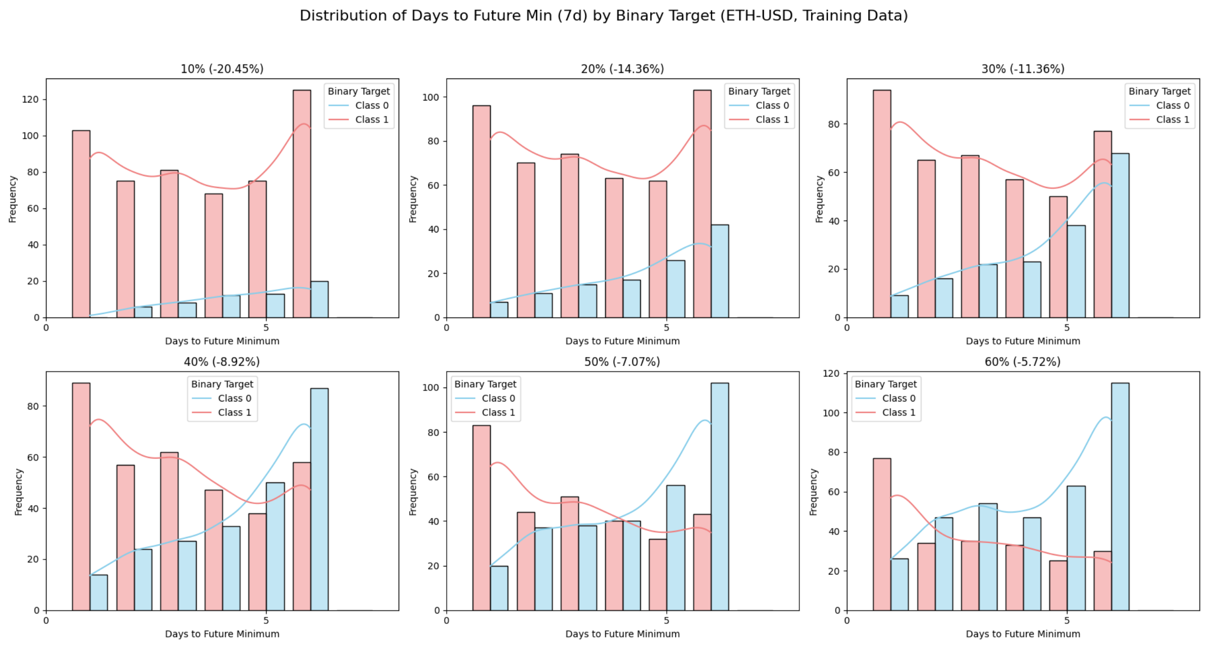

Figure 5: Distribution of Days to Future Minimum ($Y_{reg,t}$) for instances where $Y_{bin,t} = 1$, under varying T_{pct} thresholds (Training Data).

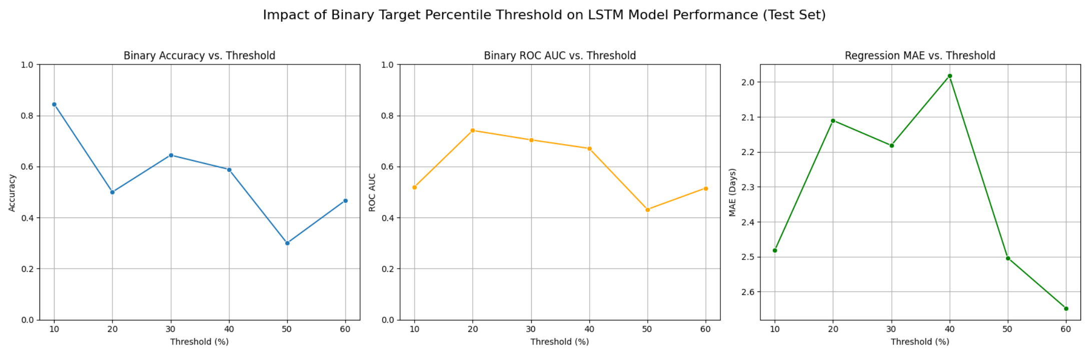

Figure 6: Preliminary evaluation of model performance metrics (Binary Accuracy, ROC AUC, Regression MAE) for different T_{pct} thresholds using a baseline LSTM model.

2.3 Predictor Variable Construction

With the target variables defined, the next step was to construct the predictor variables (features) that the deep learning models would use to make predictions. This involved creating lagged versions of the primary cryptocurrency's own data and incorporating lagged values of the external factors. This approach aligns with ISLP's guidance on constructing non-linear and time-shifted predictors [15].

2.3.1 Self-Lags from Primary Cryptocurrency (Ethereum)

Historical price and volume data of Ethereum itself are expected to be strong predictors of its future movements. We generated time-shifted (lagged) versions of Ethereum's daily `close` price. Specifically, for an observation at time t, we included N_{self} previous closing prices as features: $P_{t-1}, P_{t-2}, \ldots, P_{t-N_{self}}$. In this study, `SELF_LAG_NUM` was set to 30, creating 30 lagged price features. These self-lags were generated independently for the training, validation, and test sets after the initial chronological split to prevent any look-ahead bias.

2.3.2 Lags from External Factors

To provide the models with a richer market context, lagged values from the external data sources (described in Section 2.1.2) were also incorporated as features. This included:

- **Bitcoin Lags:** Lagged `close` prices of Bitcoin (BTC-USD).
- **FRED Data Lags:** Lagged values of the 10-Year U.S. Treasury Constant Maturity Rate.
- **Fear & Greed Index Lags:** Lagged values of the Crypto Fear & Greed Index.
- **S&P 500 Lags:** Lagged `close` prices of the S&P 500 index.

The number of lags created for these external factors was all set to 30 days. As mentioned, these external factors and their lags were initially processed on the full dataset before being merged with the date-corresponding observations in the already split Ethereum data. This is methodologically sound as these external series are considered exogenous to Ethereum's price at any given t.

2.4 Data Splitting and Leakage Prevention Strategy

Preventing data leakage—where information from outside the training data is inadvertently used to create the model, leading to overly optimistic performance estimates—is paramount in time series forecasting. We employed several robust strategies.

2.4.1 Chronological Train-Validation-Test Split

The most fundamental safeguard is strict chronological splitting of the data for Ethereum. The dataset was divided into three distinct, non-overlapping periods:

- **Training Set:** The oldest contiguous block of data, used exclusively for model parameter learning.
- **Validation Set:** The block of data immediately following the training set, used for hyperparameter tuning, model selection, and early stopping.
- **Test Set:** The most recent block of data, held out entirely until final model evaluation to provide an unbiased estimate of generalization performance on unseen data.

This chronological partitioning, forming a hold-out validation and test set strategy (ISLP, Chapter 5.3) [15], ensures that the model is trained on past data to predict future, yet-unseen periods, mimicking a real-world forecasting scenario. Figure 7 illustrates this partitioning. To make the sets reasonably similar, the splits were made so that each set included at least one "hype" period where Etherium rose quickly for some time, and one "panic" period where Etherium lost a lot of value quickly.

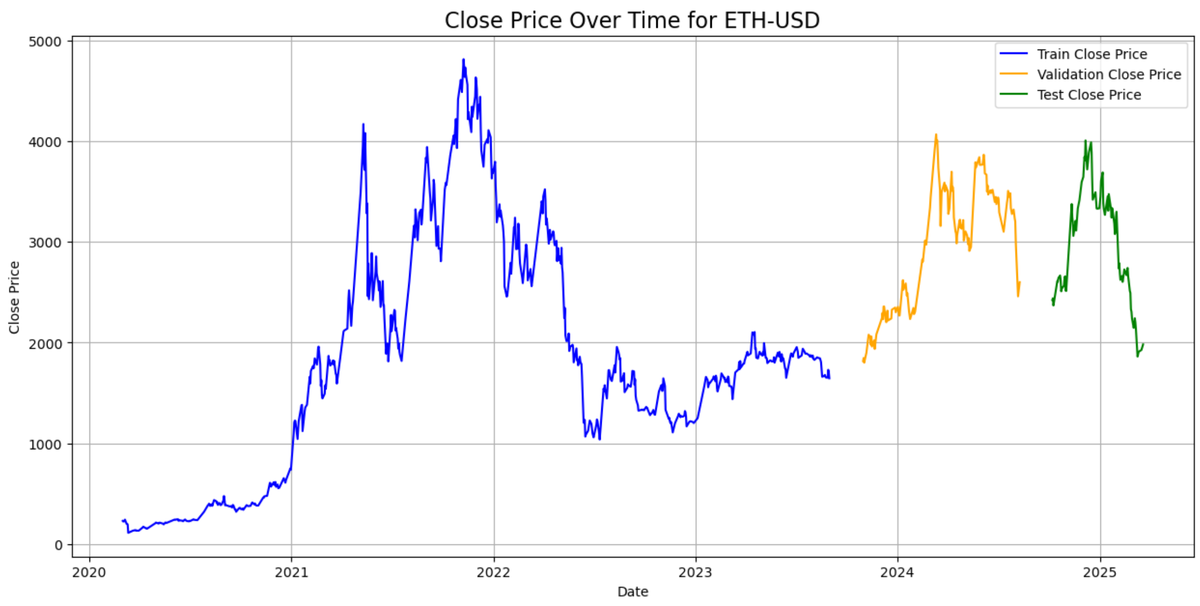

Figure 7: Illustration of Chronological Data Splitting into Training, Validation, and Test Sets with Buffer Periods.

2.4.2 Buffer Periods

To further mitigate the risk of information leakage across split boundaries, especially when using features that involve rolling calculations or lags that could span the split points, a buffer period (`BUFFER_DAYS`, e.g., 30 days) was introduced between the training and validation sets, and between the validation and test sets. Data points falling within these buffer periods were excluded from all sets, creating a clearer separation.

2.4.3 Handling of NaN Values

The process of generating lagged features inevitably introduces Not-a-Number (NaN) values at the beginning of each data series (or split). For instance, creating N lags results in the first N observations having incomplete feature sets. All rows containing any NaN values were removed (`dropna()`) from the training, validation, and test sets independently after all feature engineering and splitting steps. This ensures that models are only trained and evaluated on complete, valid observations.

3 Model Development

The successful prediction of future cryptocurrency price lows, particularly for a volatile asset like Ethereum (ETH-USD), hinges on a robust model development pipeline and a rigorous training methodology. This section details this pipeline, beginning with the final data preparation steps necessary to transform engineered features into a format suitable for deep learning models. It then provides a comprehensive exposition of the diverse neural network architectures investigated in this study. Finally, it delineates the common training, validation, and regularization strategies employed to ensure fair comparison and to foster models that generalize well to unseen data. Our approach leverages established deep learning techniques while systematically exploring various architectures to capture the complex temporal dynamics inherent in financial markets.

3.1 Final Data Preparation for Modeling

Building upon the data acquisition, feature engineering, and chronological splitting strategies detailed in Section 2, the partitioned datasets (training, validation, and test) for ETH-USD underwent final transformations to be directly consumable by the neural network models.

3.1.1 Feature Scaling

Neural networks, particularly those using gradient-based optimization, often benefit from feature scaling, which helps to normalize the range of input variables and can lead to faster convergence and more stable training. All numeric predictor variables selected for modeling (as constructed in Section 2.3) were scaled to the interval [0, 1] using the `MinMaxScaler` from the Scikit-learn library[21]. Crucially, to prevent data leakage, the scaler was parameterized (i.e., its 'fit' method was called) exclusively using summary statistics (minimum and maximum values for each feature) derived from the *training data partition only*. This fitted scaler was then subsequently used to transform (i.e., apply the 'transform' method) the corresponding features in the validation and test partitions. This ensures that no information about the data distribution of the validation or test sets influences the scaling process applied to the training data or to the subsequent sets themselves, a best practice outlined in ISLP (Chapter 2, Chapter 10) [15].

3.1.2 Sequential Data Restructuring (for RNNs/TCNs)

To enable recurrent neural networks (LSTMs, GRUs) and temporal convolutional networks (TCNs) to effectively learn from historical patterns, the scaled feature data were transformed into sequences of fixed length. A utility function, `create_sequences`, was implemented to generate input sequences of the form $(X_{t-\texttt{TIME\_STEPS}+1}, \dots, X_t)$, where each X_i denotes the vector of scaled features at time i.

In this study, `TIME_STEPS` was set to 10 days. This window length was chosen to provide sufficient historical context for daily predictions while managing model complexity and computational cost. Each generated input sequence was paired with its corresponding target variables (`binary_target` and `days_to_future_min_7d`, as defined in Section 2.2) for the immediately following time step, $t + 1$.

This sequential transformation was applied independently to the training, validation, and, eventually, test data partitions. The resultant input tensor shape for the sequence-based models was therefore $(batch\_size, 10, number\_of\_features)$. For non-sequential models such as a standard MLP, the input was reshaped or directly used as $(batch\_size, number\_of\_features\_in\_window)$, where lags were flattened.

3.2 Neural Network Architectures

A portfolio of deep learning architectures was designed and implemented using Keras [7, 8] with a TensorFlow backend[1], addressing the dual-output prediction task (binary classification and regression). This comparative approach is motivated by the principle of model selection (ISLP, Chapter 5.4) [15], allowing for an empirical evaluation of different strategies for capturing temporal dependencies and feature interactions. All models culminated in two distinct output layers:

1. A dense layer with a single neuron and a sigmoid activation function ($\sigma(x) = 1/(1 + e^{-x})$) for the `binary_output`, predicting the probability of a significant price drop.
2. A dense layer with a single neuron and a linear activation function for the `regression_output`, predicting `days_to_future_min_7d`.

To combat overfitting and encourage simpler weight solutions, L2 kernel regularization (with $\lambda = 0.001$) was applied to dense layers in some architectures. Explicit L2 regularization was not added to the primary recurrent layers (LSTM, GRU) in the custom-built models shown below, though underlying libraries or specialized blocks like TCN might incorporate forms of regularization internally, consistent with regularization techniques discussed in ISLP (Chapter 6.2) [15]. The specific architectures explored are detailed below:

3.2.1 Long Short-Term Memory (LSTM) Network

RNN models tend to struggle with handling long-term dependencies in sequential data, as the gradients are diminishing when they are backpropogated through the network, a phenomenon known as the vanishing gradient problem [26]. LSTMs are specifically designed to mitigate this problem by using memory gates that control the inflow and outflow of information that is retained in each memory cell [3][19]. Predicting future prices of financial instruments based on recent time-series data for a set of explanatory variables, require a model that is adept at learning sequential data. Thus, the first model evaluated in this task is an LSTM model.

The model features a stacked architecture, which starts off with three LSTM layers (256-128-64 units) using 'tanh' activation functions. The next step is a Dense layer with 64 units and ReLU activation, before the two output heads. The regression output (predicted days until lowest price) has 1 unit and linear activation, and the binary output (predicted price above or below threshold) with 1 unit and sigmoid activation. In between all of these layers, there are dropout layers with 20% dropout rate for regularization.

This structure is designed to gradually extract and improve time-related features from the input sequences.

3.2.2 Gated Recurrent Unit (GRU) Network

GRU is another, more recent type of RNN, with similar traits as the LSTM model, but with different gates to control the inflow and outflow of information to the memory cells that in theory should make training faster with similar accuracy as LSTM models [20] [24]. The GRU model structure and parameters are identical to the LSTM model with the only difference being that the LSTM layers have been replaced by GRU layers.

3.2.3 Hybrid LSTM-GRU Network

This architecture explores synergistic learning by processing the input sequence through parallel LSTM and GRU pathways, whose outputs are then combined:

- *Input Layer:* Splits or feeds the same input to two branches.
- *LSTM Pathway:* An LSTM layer (128 units, `return_sequences=True`), Dropout (0.3), followed by another LSTM layer (64 units, `return_sequences=False`) and Dropout (0.3).
- *GRU Pathway:* A GRU layer (128 units, `return_sequences=True`), Dropout (0.3), followed by another GRU layer (64 units, `return_sequences=False`) and Dropout (0.3).
- The final outputs (not sequences, but the final hidden state) from both pathways are concatenated.
- This concatenated representation is then processed by a Dense layer (64 units, ReLU activation) with Dropout (0.3) before the output heads.

The rationale is that LSTMs and GRUs might capture subtly different temporal patterns, and their combination could yield a richer, more robust feature set for the final prediction.

3.2.4 Temporal Convolutional Network (TCN)

Temporal Convolutional Networks (TCNs), introduced by Bai et al. (2018) [6], are a type of neural network specifically designed for sequence data. They combine the strengths of convolutional neural networks (CNNs), such as efficient parallel processing and stable gradients, with the ability to capture long-range dependencies in time series, a domain traditionally held by recurrent neural networks (RNNs) like LSTMs and GRUs. TCNs often outperform RNNs on many sequence modeling tasks. The core TCN architecture is built on three main principles: causal convolutions, dilated convolutions, and the use of residual blocks.

Causal Convolutions To predict something in a time series (e.g., a price tomorrow), we can only use information from the past and present, not the future. TCNs ensure this through "causal convolutions." This means that when the network calculates a value for a time step t, it only uses input data from time step t and earlier time steps $(x_0, \ldots, x_t)$. This prevents the model from "cheating" by looking at future data it shouldn't have access to.

Dilated Convolutions To effectively look far back into the time series without needing an extremely deep network or very large filters, TCNs employ "dilated convolutions." Imagine a filter that typically looks at consecutive data points. A dilated convolution skips a certain number of data points between each point the filter examines. By increasing this "skip factor" (dilation factor) in deeper layers, the network can achieve a very large "receptive field"—that is, the ability to capture relationships over long time periods—with fewer layers and parameters than standard convolutions would require. This makes them efficient for modeling long historical contexts.

Residual Blocks TCNs often stack many layers of dilated causal convolutions to build deep networks. To make the training of such deep networks easier and more effective (and avoid issues like vanishing gradients), TCNs use "residual blocks," inspired by ResNet architectures [14]. In a residual block, the input data is not only passed through the convolutional layers in the block but is also added directly to the block's output via a "skip connection." This aids information flow and gradients through the network, allowing for the construction of deeper and more robust models. Each residual block typically contains two sets of dilated causal convolutions, normalization, an activation function (e.g., ReLU), and dropout for regularization.

Advantages over RNNs TCNs offer several advantages compared to traditional RNNs:

- **Parallelism:** Convolutions can be computed in parallel for all time steps within a layer, unlike the sequential processing in RNNs. This can lead to significant speedups during training.
- **Flexible and Large Receptive Field:** The size of the receptive field can be easily controlled by adjusting the number of layers, filter size, and dilation factors, allowing for the modeling of very long dependencies.
- **Stable Gradients:** TCNs are generally less susceptible to the vanishing or exploding gradient problems that often plague RNNs, especially for long sequences.
- **Lower Memory Requirement for Training Long Sequences:** For very long sequences, TCNs can have lower memory requirements during training compared to standard RNNs.

Leveraging these architectural strengths, our TCN architecture for predicting cryptocurrency price lows comprised:

- A TCN block (using a library like `keras-tcn` or a custom implementation) with, for example, 128 filters, a kernel size of 2, and a stack of dilated convolutions with increasing dilation rates (e.g., [1, 2, 4, 8, 16]). Key TCN features such as causal padding (to ensure no future peeking), skip connections (inherent in residual blocks), and potentially layer normalization were employed. A dropout rate (e.g., 0.3) was applied within the TCN block for regularization. `return_sequences=False` was used as the TCN block's output was directly fed to dense layers for feature aggregation.
- The output vector from the TCN block was then passed through a stack of Dense layers (e.g., 512, 256, 128, 64, and 32 units, each with ReLU activation and dropout, e.g., 0.3) to progressively refine features and reduce dimensionality before the final prediction heads.

3.2.5 Comprehensive Hybrid TCN-LSTM-GRU Network

This model represents an advanced ensemble-like approach, aiming to leverage the distinct strengths of TCNs, LSTMs, and GRUs simultaneously through parallel processing streams which are then aggregated:

- *Input Layer:* Feeds the same input sequence to three parallel streams.

- *LSTM Stream:* An LSTM layer (e.g., 128 units, `return_sequences=False`, dropout 0.1, recurrent dropout 0.1), potentially followed by Dense layers (e.g., 64 units then 16 units, each with ReLU and dropout 0.1).
- *GRU Stream:* Structured similarly to the LSTM stream, but using GRU layers.
- *TCN Stream:* A TCN block (e.g., 128 filters, configured as in the standalone TCN model, `return_sequences=False`), potentially followed by Dense layers (e.g., 64 units then 16 units, each with ReLU and dropout 0.1).
- The final (non-sequential) outputs from these three streams are concatenated.
- This aggregated feature vector is then processed by further Dense layers (e.g., 128, 64, 32 units, all with ReLU and dropout 0.1) before the final output heads.

3.2.6 Hybrid LSTM-GRU with Multi-Head Self-Attention

Inspired by the Transformer architecture [28], this model incorporates a self-attention mechanism to allow the network to dynamically weigh the importance of different time steps within the input sequences when forming a representation.

- *Parallel Recurrent Encoders:* The input sequence is processed by two parallel branches:
  - LSTM Branch: A stack of LSTM layers (e.g., three layers: 64, 32, then 16 units; all `return_sequences=True` to output the full sequence for attention, L2 kernel regularization, dropout 0.3).
  - GRU Branch: A stack of GRU layers (e.g., three layers: 64, 32, then 16 units; all `return_sequences=True`, L2 kernel regularization, dropout 0.3).
- The sequence outputs from these recurrent branches are concatenated along the feature dimension.
- *Transformer Encoder Block:* This combined sequence (now with richer features per time step) is processed by a custom `transformer_encoder` block. This block typically implements multi-head self-attention (e.g., 2 heads, key dimension of 32 per head) followed by a position-wise feed-forward network, with residual connections and layer normalization applied after each sub-layer.
- *Context Aggregation:* A Global Average Pooling layer (or Global Max Pooling, or Flatten) is applied to the output sequence of the attention mechanism to produce a fixed-size context vector. This vector is then passed through a Dense layer (64 units, ReLU activation, dropout 0.3) before being fed to the output heads.

3.3 Model Training and Validation

A consistent and rigorous training protocol was applied across all developed model architectures to ensure fair comparison and promote robust learning.

3.3.1 Optimizer and Learning Rate

The Adam (Adaptive Moment Estimation) optimizer was employed for gradient-based optimization of the model weights. Adam was chosen due to its efficient nature and being "well suited for problems that are large in terms of data and/or parameters" (Kingma & Ba, 2014) [10]. It was configured with an initial learning rate of 0.001 (and default Keras values for other hyperparameters ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$) [27].

3.3.2 Loss Function Design (Multi-task Learning)

Given the dual-output nature of our prediction task (binary classification and continuous regression), a composite loss function was defined. The total loss to be minimized during training was a weighted sum of the individual losses for each output head:

- For the `binary_output` (classification task): `binary_crossentropy` loss was used, which is appropriate for probabilistic binary outputs from a sigmoid activation.
- For the `regression_output` (regression task): `mean_absolute_error` (MAE) was used as the loss function to encourage the model to produce varied predictions rather than consistently converging to the mean expected value.
  In contrast, using `mean_squared_error` (MSE) would penalize larger prediction errors more heavily due to its quadratic nature, making the model more likely to favor predictions closer to the expected value and less tolerant of deviations.

Initially, experiments with using different weighting schemes for the losses were conducted. For example: `loss_weights = {'binary_output': 1.0, 'regression_output': 0.5}`, to place a slightly greater emphasis on the binary classification task. The rationale would be that correctly identifying the potential for a significant drop is of primary practical importance, with the exact timing (regression task) being a valuable, albeit secondary, information. However, the asymmetrical losses were dropped in the final model, as it only resulted in uniform predictions very close to the expected values. Further exploration of asymmetrical loss weights could be an avenue for future optimization.

3.3.3 Performance Evaluation Metrics

During training (monitored on the validation set) and for final model evaluation (on the test set), a suite of metrics was tracked to comprehensively assess performance on both tasks:

- For the `binary_output` (Classification):
  - **Accuracy**: The proportion of correct classifications.
  - **Precision**: $\mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$, measuring the accuracy of positive predictions.
  - **Recall (Sensitivity)**: $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, measuring the ability to identify actual positive cases.
  - **F1-Score**: $2\times(\mathrm{Precision}\times\mathrm{Recall})/(\mathrm{Precision}+\mathrm{Recall})$, the harmonic mean of precision and recall.

- **ROC AUC (Area Under the Receiver Operating Characteristic Curve)**: Measures the model's ability to distinguish between the positive and negative classes across all classification thresholds. An ROC curve plots True Positive Rate (Sensitivity) against False Positive Rate (1 - Specificity), as illustrated generically in Figure 8. A higher AUC indicates better discrimination.

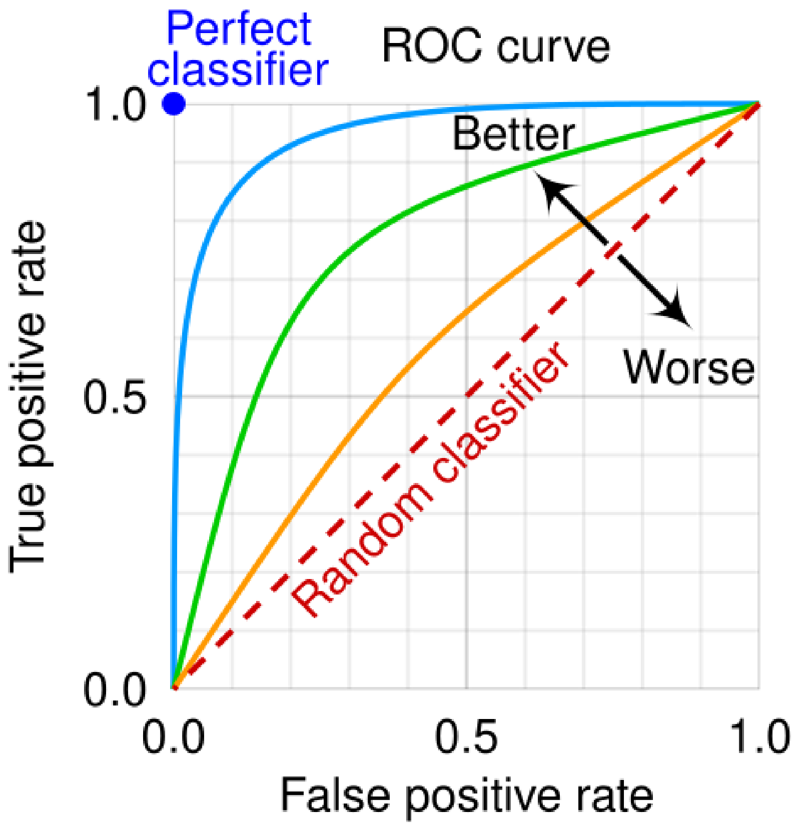

Figure 8: Generic illustration of ROC curve space, showing the trade-off between True Positive Rate and False Positive Rate. The diagonal line represents a random classifier (AUC=0.5).

- For the `regression_output` (Regression):
  - **Mean Absolute Error (MAE)**: $(1/n)\sum_{i=1}^{n}|y_i - \hat{y}_i|$, providing an interpretable measure of the average prediction error in the original unit (days).
  - **Root Mean Squared Error (RMSE)**: $\sqrt{(1/n)\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$, which penalizes larger errors more significantly than MAE.

3.3.4 Training Configuration (Epochs, Batch Size)

Models were trained for a maximum of `EPOCHS` = 2000 epochs. Data was fed to the models in mini-batches of size `BATCH_SIZE` = 8. The choice of batch size can affect training stability, generalization, and training speed. The relatively low batch size was chosen due to the emphasis on accuracy rather than efficiency in training [16] [18] and is a power of 2, as this optimizes the matrix operations in the computer when loading and training on the data[23]. The number of epochs was set so that the training time had a reasonable ceiling.

3.3.5 Overfitting Mitigation and Callbacks

Several techniques were employed to combat overfitting and improve the generalization capability of the models:

- **Dropout:** Dropout layers (with rates of 0.3, as specified in the architecture descriptions) were inserted between layers to randomly deactivate a fraction of neurons during training, preventing co-adaptation of neurons.

- **Early Stopping:** To prevent overfitting by stopping training when performance on unseen data ceases to improve, Keras' `EarlyStopping` callback was implemented. It monitored the total validation loss (`val_loss`) and was configured to stop training if no improvement (defined by a minimum delta, `min_delta`, e.g., 0.0001) was observed for a specified number of `patience` epochs. `PATIENCE_EARLY_STOPPING` was set to 200 consecutive epochs, allowing the model to explore different weights extensively, but within reasonable limits, before stopping and settling for the best fit to the validation set it found.

- **Reduce Learning Rate On Plateau:** Another callback from the Keras library, `ReduceLROnPlateau`, was utilized to reduce the learning rate when the training plateaus. Similarly to `EarlyStopping` it monitored the total validation loss, and with `factor` set to 0.5 and `patience` to 90, the models' learning rate is halved twice before `EarlyStopping` kicks in. This allows the model to search extensively for small improvements before giving up, potentially yielding better predictions.

- **Restore Best Weights:** Critically, the `restore_best_weights=True` parameter of the `EarlyStopping` callback was enabled. This ensures that the final model weights loaded after training are those corresponding to the epoch that yielded the minimum validation loss, rather than the weights from the last training epoch. This is a key mechanism for selecting the model state that demonstrated the best generalization capability on the validation data.

3.3.6 Reproducibility

To promote the reproducibility of the experimental results, random seeds were set for Python's built-in `random` module, NumPy (`np.random.seed()`), and TensorFlow (`tf.random.set_seed()`) at the beginning of each experimental run. While perfect reproducibility can be challenging in complex deep learning pipelines, especially with GPU parallelism, this helps to minimize stochastic variations.

4 Results

This section presents the empirical outcomes from training and evaluating the various deep learning architectures detailed in Section 3.2. The primary focus is on the performance of these models on the held-out validation dataset for Ethereum (ETH-USD). The presentation herein is factual, reporting the observed training dynamics and quantitative performance metrics. Interpretive commentary and broader implications are reserved for Section 5.

4.1 Experimental Setup Overview

The experiments involved training and evaluating six distinct deep learning architectures:

- LSTM Model
- GRU Model
- Hybrid LSTM-GRU Model
- TCN Model
- Comprehensive Hybrid TCN-LSTM-GRU Model
- Hybrid LSTM-GRU Model with Multi-Head Self-Attention

All models were trained on the ETH-USD training dataset, which spanned from 1.Jan 2020 to 31.Aug 2023, following the data preparation steps outlined in Section 3.1. Model performance was primarily assessed on the validation set, covering the period from 31.Oct 2023 to 9.Aug 2024, representing approximately 9 months of unseen data (11 is specified in the code, but 2 months at both ends are lost in buffer zones to avoid data leakage). The target variables were the 7-day binary drop indicator (using a T_{pct} threshold of the 40th percentile of price change, corresponding to $\approx$ -7.08%) and the regression target for days to minimum, as defined in Section 2.2. The 40th percentile was chosen to achieve a balanced distribution of the binary target in the training data, ensuring the models have sufficient data points for both classes to train effectively. Key hyperparameters for training, such as learning rate (0.001), batch size (8), and maximum epochs (2000), were kept consistent across models as specified in Section 3.3, with early stopping employed.

4.2 Model Performance: LSTM Network

4.2.1 Training Dynamics and Convergence

The LSTM model was trained on ETH-USD sequences, with early stopping monitoring the combined validation loss. Training concluded after 258 epochs, with the lowest validation loss achieved at epoch 58. Figure 9 illustrates the training and validation loss curves (total, binary, and regression losses). Training loss decreased consistently, but validation loss trended slightly upward, indicating overfitting, particularly for the binary output.

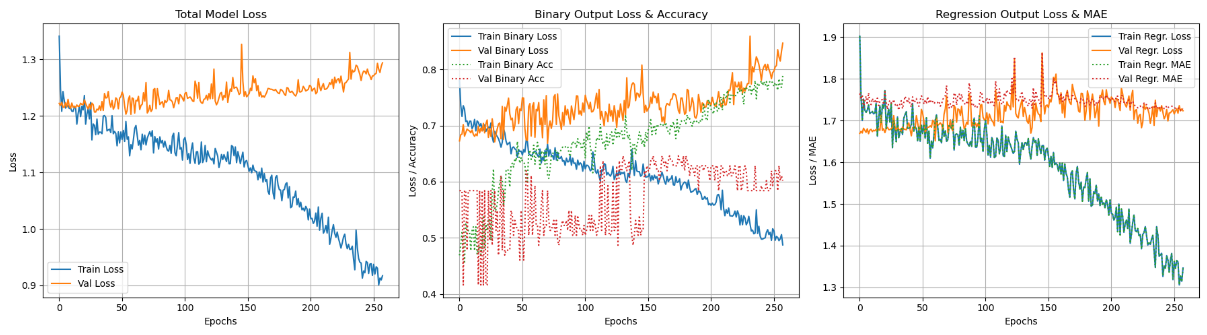

Figure 9: Training and Validation Loss Curves (Total Loss, Binary Loss & Regression Loss) for the LSTM Model on ETH-USD Data. Lowest validation loss at epoch 58.

4.2.2 Test Set Performance

The LSTM model's performance on the test set, with weights restored from the best validation epoch, is summarized in Table 2. Despite seemingly promising metrics, further analysis (Figures 10 and 11) reveals that the model predicted the positive class (drop) and approximately 3.8 days for all test data points, resembling a naive baseline predicting the majority class and expected regression value.

Table 2: Quantitative Performance Metrics of the Trained LSTM Model on the ETH-USD Test Set.

Performance Metric	**Achieved Value**
Binary Classification (`binary_target`)	
Accuracy	0.59
Precision	0.35
Recall	0.59
F1-Score	0.44
ROC AUC	0.65
Regression (`days_to_future_min_7d`)	
Mean Absolute Error (MAE)	1.53 days
Root Mean Squared Error (RMSE)	1.77 days

Note: Binary target threshold was the 40th percentile. Metrics are for the ETH-USD test period: 7.Oct 2024 - 20.Mar 2025.

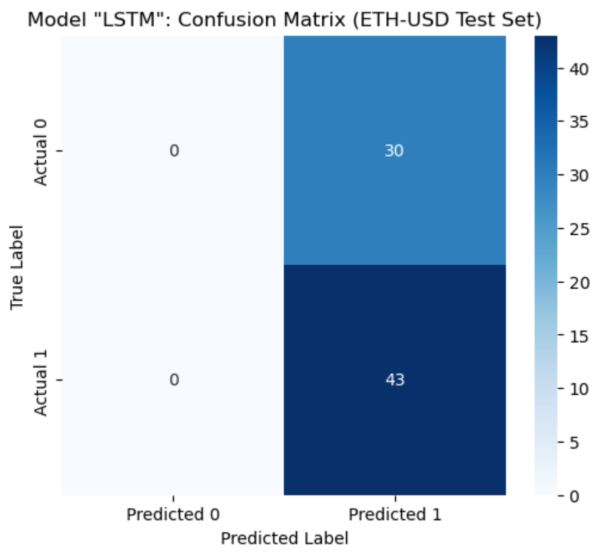

Figure 10: Confusion Matrix of the Binary Classification for the LSTM Model on the test data

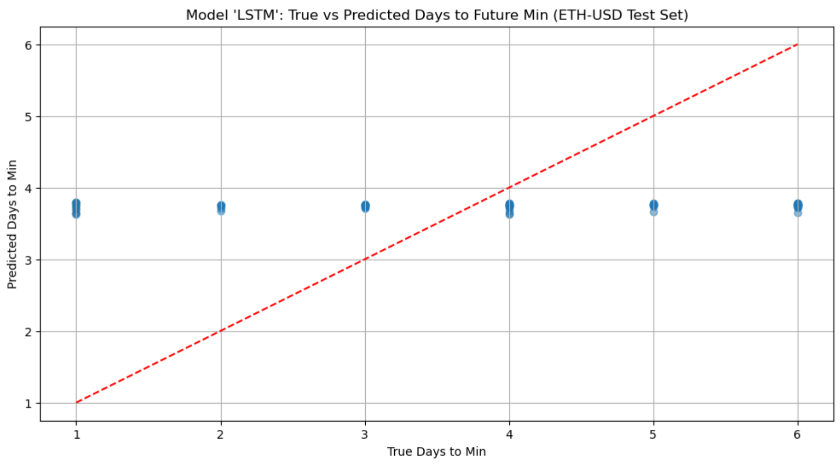

Figure 11: Predicted vs Actual days until the lowest price for the LSTM Model on the test data

4.3 Model Performance: GRU Network

4.3.1 Training Dynamics and Convergence

The GRU model converged more rapidly with the lowest validation loss at epoch 45. Figure 12 shows consistent improvement in training loss, but validation loss remained stable until epoch 220, after which overfitting occurred, primarily on the binary classification task.

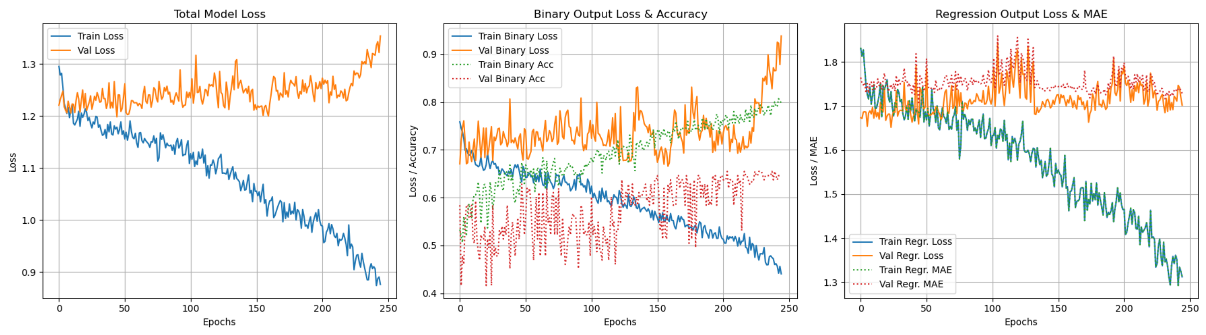

Figure 12: Training and Validation Loss Curves (Total Loss, Binary Loss & Regression Loss) for the GRU Model on ETH-USD Data. Lowest validation loss at epoch 45.

4.3.2 Test Set Performance

Table 3 presents the GRU model's test set performance, which closely mirrors the LSTM results. Figures 13 and 14 confirm that the model predicted the positive class and approximately 3.8 days uniformly, yielding metrics nearly identical to those of the LSTM model (e.g., Accuracy: 0.59, MAE: 1.50 days).

Table 3: Quantitative Performance Metrics of the Trained GRU Model on the ETH-USD Test Set.

Performance Metric	**Achieved Value**
Binary Classification (`binary_target`)	
Accuracy	0.59
Precision	0.35
Recall	0.59
F1-Score	0.44
ROC AUC	0.66
Regression (`days_to_future_min_7d`)	
Mean Absolute Error (MAE)	1.50 days
Root Mean Squared Error (RMSE)	1.78 days

Note: Binary target threshold was the 40th percentile. Metrics are for the ETH-USD test period: 7.Oct 2024 - 20.Mar 2025.

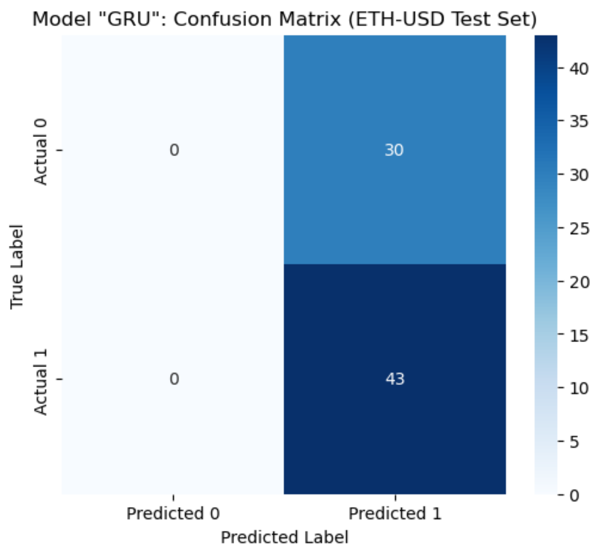

Figure 13: Confusion Matrix of the Binary Classification for the GRU Model on the test data

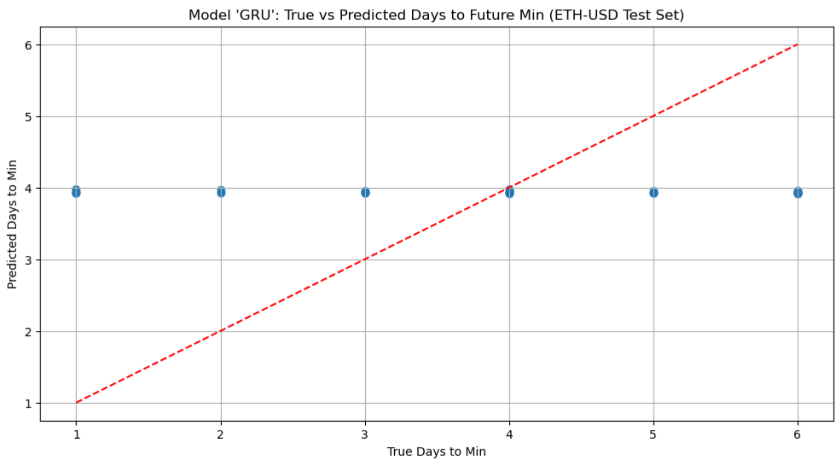

Figure 14: Predicted vs Actual days until the lowest price for the GRU Model on the test data

4.4 Model Performance: Hybrid LSTM-GRU Network

4.4.1 Training Dynamics and Convergence

The Hybrid LSTM-GRU model reached its lowest validation loss after 293 epochs. Figure 15 shows steady training loss improvement but a stable validation loss with occasional fluctuations, suggesting random variations rather than consistent learning on unseen data.

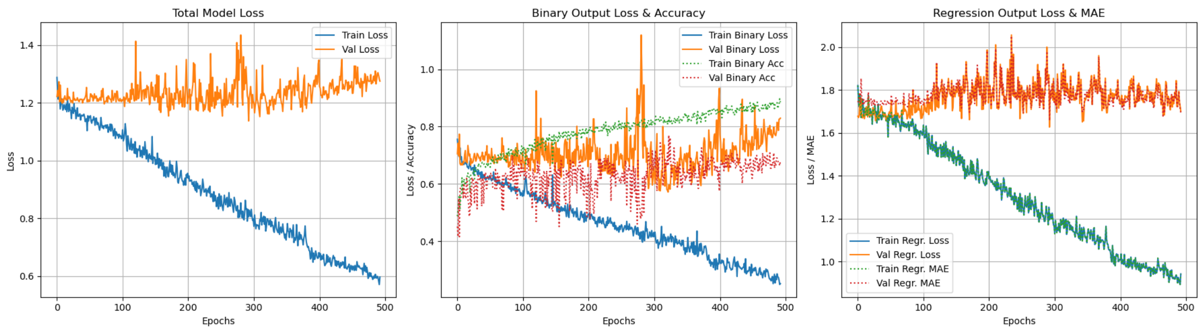

Figure 15: Training and Validation Loss Curves (Total Loss, Binary Loss & Regression Loss) for the Hybrid LSTM & GRU Model on ETH-USD Data. Lowest validation loss at epoch 293.

4.4.2 Test Set Performance

Table 4 indicates poorer performance compared to the LSTM and GRU models, except for a slightly higher precision (0.46 vs. 0.35). Figures 16 and 17 reveal a wider spread of predictions, confirming less accurate binary and regression outputs (e.g., Accuracy: 0.44, MAE: 1.62 days).

Table 4: Quantitative Performance Metrics of the Trained Hybrid LSTM-GRU Model on the ETH-USD Test Set.

Performance Metric	**Achieved Value**
Binary Classification (`binary_target`)	
Accuracy	0.44
Precision	0.46
Recall	0.44
F1-Score	0.44
ROC AUC	0.44
Regression (`days_to_future_min_7d`)	
Mean Absolute Error (MAE)	1.62 days
Root Mean Squared Error (RMSE)	1.91 days

Note: Binary target threshold was the 40th percentile. Metrics are for the ETH-USD test period: 7.Oct 2024 - 20.Mar 2025.

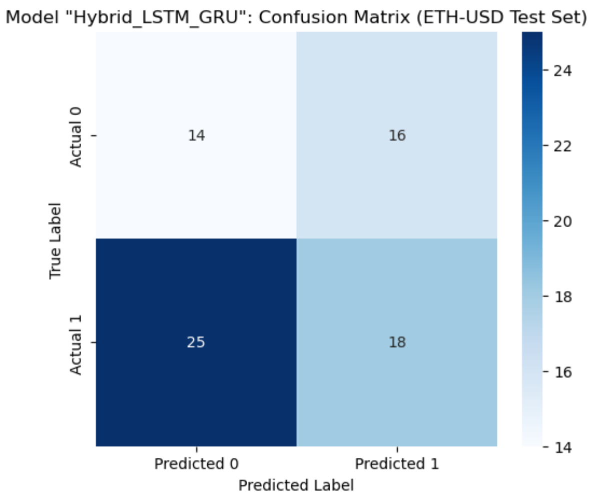

Figure 16: Confusion Matrix of the Binary Classification for the Hybrid LSTM & GRU Model on the test data

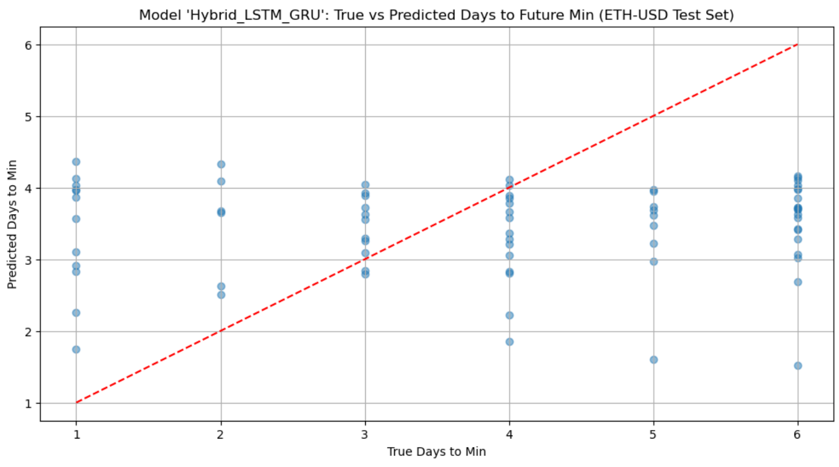

Figure 17: Predicted vs Actual days until the lowest price for the Hybrid LSTM & GRU Model on the test data

4.5 Model Performance: TCN

4.5.1 Training Dynamics and Convergence

The TCN model exhibited minimal learning on the validation set after approximately 30 epochs, as shown in Figure 18. Training loss also showed no improvement, suggesting the model struggled to capture meaningful patterns.

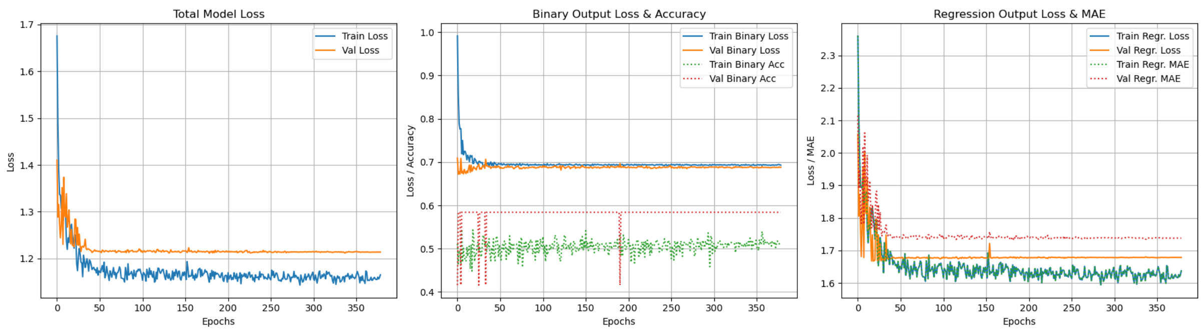

Figure 18: Training and Validation Loss Curves (Total Loss, Binary Loss & Regression Loss) for the TCN Model on ETH-USD Data. Lowest validation loss at epoch 179.

4.5.2 Test Set Performance

Table 5 confirms the TCN model's poor performance, with the lowest metrics across all models (e.g., Accuracy: 0.41, ROC AUC: 0.38). Figures 19 and 20 indicate uniform predictions similar to those of the LSTM and GRU models.

Table 5: Quantitative Performance Metrics of the Trained TCN Model on the ETH-USD Test Set.

Performance Metric	**Achieved Value**
Binary Classification (binary_target)	
Accuracy	0.41
Precision	0.17
Recall	0.41
F1-Score	0.24
ROC AUC	0.38
Regression (days_to_future_min_7d)	
Mean Absolute Error (MAE)	1.52 days
Root Mean Squared Error (RMSE)	1.78 days

Note: Binary target threshold was the 40th percentile. Metrics are for the ETH-USD test period: 7.Oct 2024 - 20.Mar 2025.

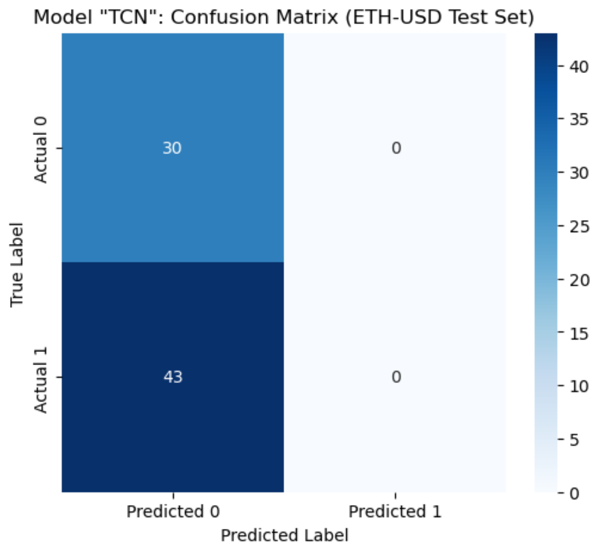

Figure 19: Confusion Matrix of the Binary Classification for the TCN Model on the test data

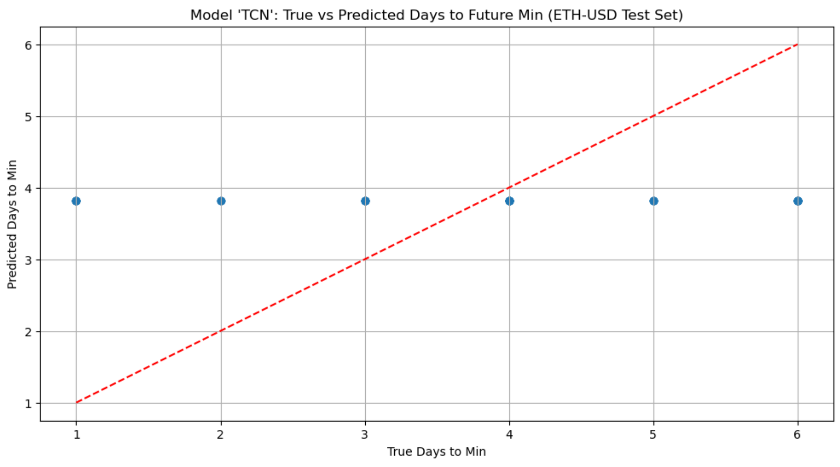

Figure 20: Predicted vs Actual days until the lowest price for the TCN Model on the test data

4.6 Model Performance: Mega TCN

4.6.1 Training Dynamics and Convergence

The Mega TCN model showed steady training loss reduction and a decline in validation loss until approximately epoch 100, after which validation loss increased, indicating overfitting (Figure 21). The lowest validation loss occurred at epoch 84.

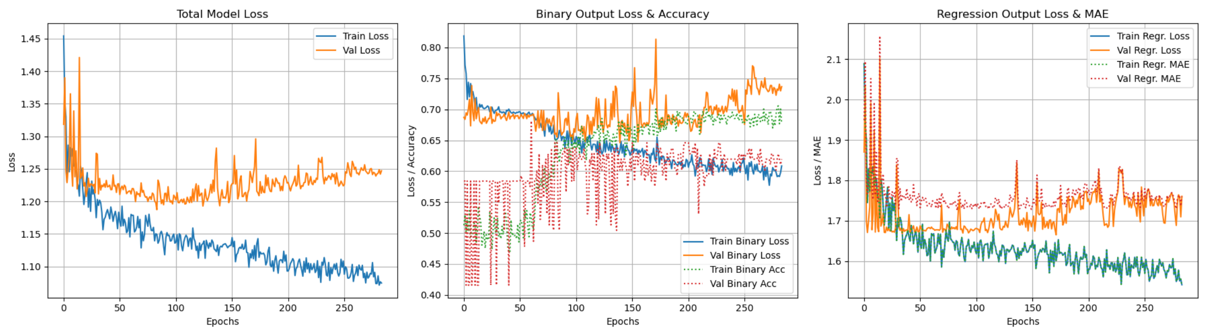

Figure 21: Training and Validation Loss Curves (Total Loss, Binary Loss & Regression Loss) for the Mega TCN Model on ETH-USD Data. Lowest validation loss at epoch 84.

4.6.2 Test Set Performance

Table 6 demonstrates that the Mega TCN model outperformed other architectures in binary classification (Accuracy: 0.63, F1-Score: 0.58, ROC AUC: 0.65). Figure 22 shows balanced predictions for both classes, but Figure 23 indicates regression predictions centered around the expected value (approximately 3.8 days), similar to other models (MAE: 1.50 days).

Table 6: Quantitative Performance Metrics of the Trained Mega TCN Model on the ETH-USD Test Set.

Performance Metric	**Achieved Value**
Binary Classification (`binary_target`)	
Accuracy	0.63
Precision	0.63
Recall	0.63
F1-Score	0.58
ROC AUC	0.65
Regression (`days_to_future_min_7d`)	
Mean Absolute Error (MAE)	1.50 days
Root Mean Squared Error (RMSE)	1.78 days

Note: Binary target threshold was the 40th percentile. Metrics are for the ETH-USD test period: 7.Oct 2024 - 20.Mar 2025.

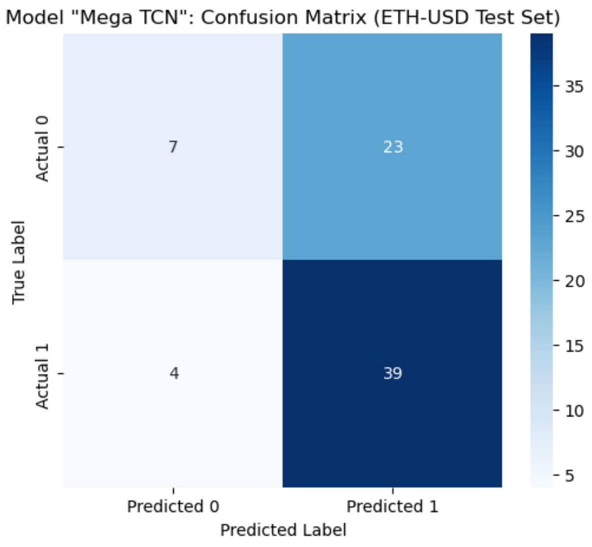

Figure 22: Confusion Matrix of the Binary Classification for the "Mega TCN" Model on the test data

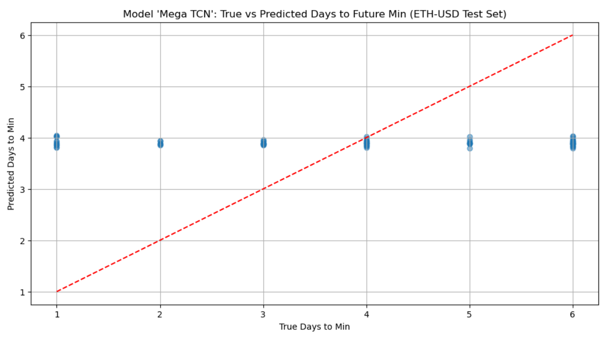

Figure 23: Predicted vs Actual days until the lowest price for the Mega TCN Model on the test data

4.7 Model Performance: "Hybridatt"

4.7.1 Training Dynamics and Convergence

The Hybridatt model improved training loss but showed no significant reduction in validation loss, converging early at epoch 18 (Figure 24). This suggests limited learning on unseen data.

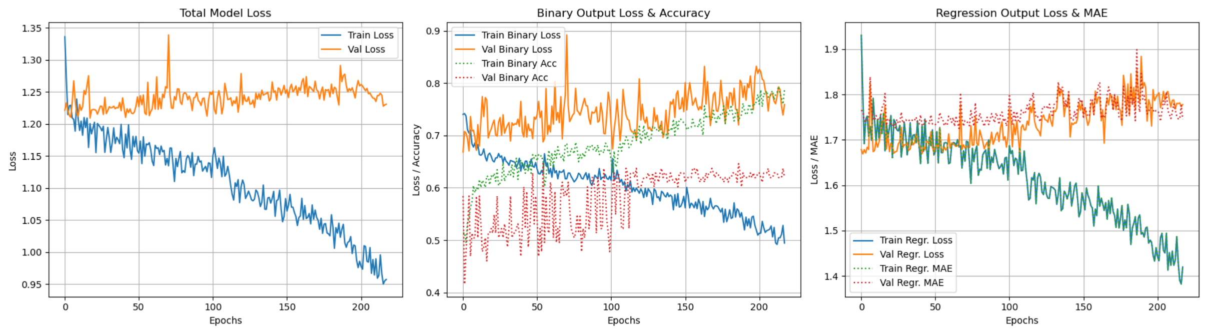

Figure 24: Training and Validation Loss Curves (Total Loss, Binary Loss & Regression Loss) for the Hybridatt Model on ETH-USD Data. Lowest validation loss at epoch 18.

4.7.2 Test Set Performance

Table 7 and Figures 25 & 26 confirm uniform predictions similar to the LSTM, GRU, and TCN models, with metrics nearly identical to the LSTM and GRU (e.g., Accuracy: 0.59, MAE: 1.51 days).

Table 7: Quantitative Performance Metrics of the Trained "Hybridatt" Model on the ETH-USD Test Set.

Performance Metric	**Achieved Value**
Binary Classification (binary_target)	
Accuracy	0.59
Precision	0.35
Recall	0.59
F1-Score	0.44
ROC AUC	0.64
Regression (days_to_future_min_7d)	
Mean Absolute Error (MAE)	1.51 days
Root Mean Squared Error (RMSE)	1.78 days

Note: Binary target threshold was the 40th percentile. Metrics are for the ETH-USD test period: 7.Oct 2024 - 20.Mar 2025.

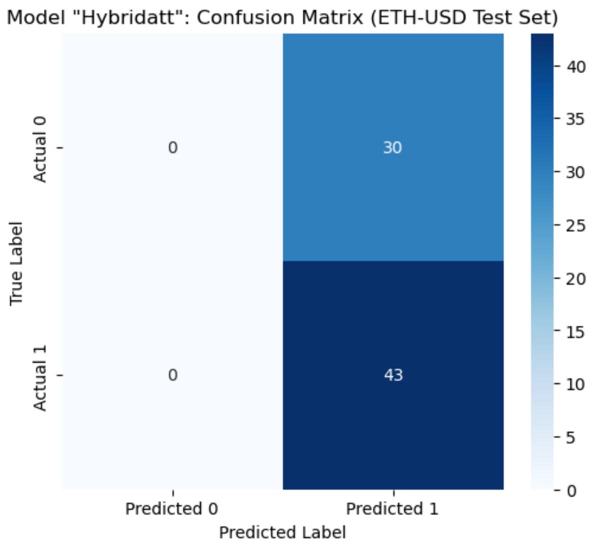

Figure 25: Confusion Matrix of the Binary Classification for the "Hybridatt" Model on the test data

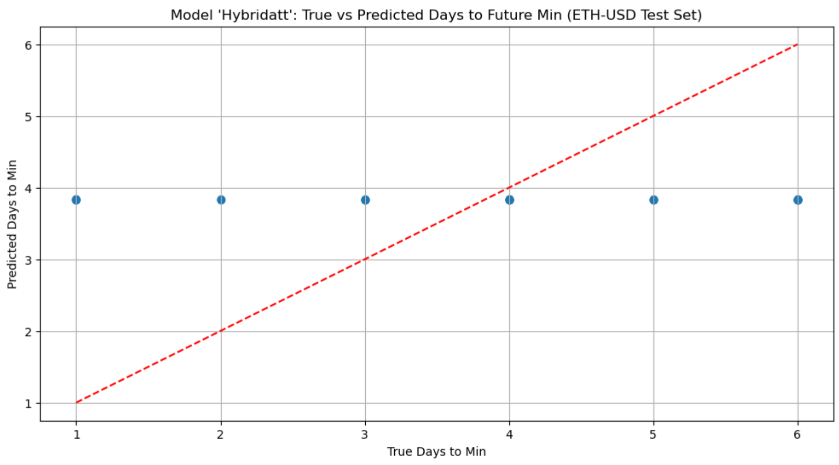

Figure 26: Predicted vs Actual days until the lowest price for the "Hybridatt" Model on the test data

5 Discussion

This section provides an interpretation of the empirical results presented in Section 4, contextualizes them within the broader field of financial forecasting using deep learning, discusses the inherent limitations of this study, and proposes potential directions for future research.

5.1 Interpretation of Key Findings

The performance of the various deep learning models on the ETH-USD validation set offers several insights into their capabilities and limitations for predicting future price lows.

5.1.1 Binary Classification Performance (Predicting Significant Drops)

The Mega TCN model achieved the highest performance for classification, with an ROC AUC of 0.65 and an F1-Score of 0.58, suggesting a moderate ability to distinguish between periods preceding significant drops versus those that do not. For instance, the Mega TCN's accuracy of 63% should be contextualized against the majority class baseline. The majority class ('0', no significant drop, based on the 40th percentile threshold) constituted approximately 60% of the validation samples. Achieving accuracy and ROC AUC values notably above this baseline (and 0.5 for ROC AUC) indicates that the models, particularly the Mega TCN, learned meaningful patterns from the historical and external features beyond random chance.

However, precision and recall scores highlight potential trade-offs. The LSTM and GRU models, while achieving a reasonable recall of 0.59, had a lower precision of 0.35, indicating they identified a good portion of actual drops but also generated a higher number of false alarms. In contrast, the Mega TCN model balanced precision (0.63) and recall (0.63), suggesting a more reliable identification of significant drops. The choice of the T_{pct} threshold at the 40th percentile influenced these metrics by defining the rarity of the positive class, ensuring a balanced dataset but potentially limiting the detection of more extreme drops.

5.1.2 Regression Task Performance (Predicting Timing of Lows)

Predicting the exact number of days until the price minimum occurs proved to be a more challenging task. The Mean Absolute Error (MAE) for the `days_to_future_min_7d` target ranged from 1.50 days for the Mega TCN and GRU models to 1.62 days for the Hybrid LSTM-GRU model. An MAE of, for example, 1.53 days (LSTM model) means that, on average, the models' prediction for when the low would occur within the 7-day window deviated by approximately 1.5 days. This represents a substantial margin of error ($1.53/7 \approx 22\%$) relative to the prediction horizon.

While these MAE values might be somewhat better than a naive strategy (e.g., always predicting the midpoint of 15 days, which would yield an MAE dependent on the true distribution), the precision is likely insufficient for applications requiring highly accurate market timing. This disparity between moderate success in classifying the occurrence of a drop and the limited precision in predicting its exact timing is a common observation in financial time series forecasting, reflecting the high noise-to-signal ratio and quasi-random characteristics of market movements.

5.1.3 Comparison Across Architectures

Section 4 reveals that the more complex Mega TCN model generally outperformed the standalone LSTM, GRU, Hybrid LSTM-GRU, TCN, and Hybridatt architectures, particularly in the binary classification task as evidenced by higher ROC AUC scores (0.65 vs. 0.38–0.66 for others) and a balanced F1-Score (0.58). Surprisingly, the simpler GRU model performed on par with the LSTM for regression, with an MAE of 1.50 days compared to 1.53 days, possibly due to its efficiency with the given dataset size and simpler architecture reducing overfitting.

For instance, the Mega TCN's superior performance in classification might be attributed to its ability to capture longer-range dependencies through dilated convolutions without the vanishing gradient issues of standard RNNs. Conversely, the basic LSTM and GRU models struggled with the extensive feature set, leading to overfitting despite regularization, as evidenced by their uniform predictions on the test set (Figures 10, 11, 13, and 14). The Hybridatt model, despite incorporating a multi-head self-attention mechanism, showed no significant improvement, with performance metrics (Table 7) nearly identical to the LSTM and GRU models, likely due to its early convergence (epoch 18) and inability to leverage the attention mechanism effectively with the given data.

The results suggest that architectures capable of handling sequential data and potentially capturing diverse temporal patterns (like the Mega TCN) offer an advantage, though the gains are incremental and come with increased computational costs. The limited performance of more complex models like Hybridatt indicates that the dataset size or feature set may not have been sufficient to justify their complexity.

5.2 Implications and Practical Considerations

The findings of this study have several implications for investors and practitioners in the cryptocurrency space. The models' moderate success in binary classification suggests they could serve as a component in a broader risk management or decision-support system. An alert indicating a heightened probability of a significant drop (based on the binary output) could prompt investors to:

- Review existing positions and consider protective measures (e.g., stop-loss orders, hedging).
- Delay new long entries until the risk signal subsides.
- Conduct further due diligence or apply other analytical methods.

The regression output, while imprecise (MAE of approximately 1.5–1.6 days), might offer a very rough temporal window for this heightened risk, but should not be relied upon for precise market timing. For example, if the binary model signals a drop and the regression model predicts the low in 5-10 days, an investor might be more cautious in that immediate timeframe rather than acting on a specific day.

However, any practical application would necessitate careful calibration of the binary threshold (T_{pct}) to align with an individual's risk appetite and the specific characteristics of the asset being traded. Furthermore, the model outputs should ideally be combined with other forms of analysis (e.g., fundamental analysis, broader market indicators not included, on-chain data) rather than being used as standalone trading signals, especially given the observed limitations in timing precision. The Efficient Market Hypothesis, in its weaker forms, suggests that past price data alone is unlikely to yield consistent alpha, a notion that these results do not strongly refute, particularly for timing.

5.3 Limitations of the Study

This research is subject to several important limitations that temper the conclusions and highlight areas for further investigation:

- **Data Granularity and Volume:** The decision to use daily data (Days) rather than higher-frequency data (such as Hours) resulted in a relatively limited number of data points. This may have constrained the models' ability to capture finer-grained, short-term patterns and potentially made them more susceptible to noise within the available daily observations.

- **Effects of Data Cutoff and Market Noise:** The delimitation of the dataset to the period from 2020 onwards, while intended to improve relevance, introduces a limitation in that earlier market regimes and potential longer cycles are not captured by the models. This could affect the models' generalization ability under market conditions more similar to the excluded periods. Furthermore, cryptocurrency markets are notoriously prone to periods of significant 'hype' and general market 'noise,' which can make it challenging for any model to discern genuine underlying signals from short-term speculative volatility. The limited historical depth post-cutoff could potentially exacerbate this challenge.

- **Feature Space Scope:** While we incorporated several external factors, the feature set could be expanded further. Potentially crucial information from intra-day price action (OHLC beyond just 'Low' for target), trading volume dynamics, more sophisticated volatility metrics (e.g., GARCH, realized volatility), established technical indicators (RSI, MACD), on-chain data (transaction volumes, active addresses, network health), and granular news sentiment analysis (e.g., using LLMs) were not included. These could significantly enhance predictive power.

- **Model Complexity vs. Data Size:** While advanced architectures were explored, their full potential might be limited by the available length of reliable historical data for ETH-USD and other cryptocurrencies. Very deep or complex models require vast amounts of data to train effectively without overfitting.

- **Fixed Target Definition and Horizon:** The 7-day prediction window and the specific percentile-based threshold for "significant drop" were chosen for this study but are not necessarily universally optimal. Different trading strategies operate on different horizons, and the definition of a significant event can vary. Sensitivity analysis to these parameters was limited.

- **Market Regime Changes and Stationarity:** Cryptocurrency markets are known for rapid evolution and distinct regime changes (e.g., bull vs. bear markets, periods of high vs. low institutional adoption). Models trained on historical data assume a degree of stationarity in underlying patterns, which may not hold true. This necessitates strategies for model adaptation or retraining.

- **Absence of Rigorous Backtesting and Transaction Costs:** The evaluation is based on predictive metrics (Accuracy, MAE, ROC AUC, etc.) on a hold-out validation set. It does not constitute a full backtest of a trading strategy incorporating transaction costs, bid-ask spreads, slippage, or capital management rules. Practical profitability or utility can only be assessed through such simulations.

- **Single Primary Asset Focus:** While external factors were included, the core predictive task and detailed model evaluation were centered on ETH-USD. The generalizability of these specific findings (e.g., which model architecture is best) to other cryptocurrencies with different market dynamics, liquidity profiles, or correlations is not guaranteed.

- **Hyperparameter Optimization:** While standard hyperparameters were used, an exhaustive systematic hyperparameter optimization (e.g., using Bayesian optimization or grid/random search via tools like Optuna or KerasTuner) for each architecture was beyond the scope of this project but could potentially yield further performance improvements.

- **Market Efficiency and Behavioral Limitations in Predicting Cryptocurrency Downturns:** The Efficient Market Hypothesis (EMH) posits that asset prices fully reflect all available information, rendering historical price patterns insufficient for consistent future predictions [11]. While our findings demonstrate that deep learning models can moderately anticipate significant price drops—suggesting weak-form inefficiency in cryptocurrency markets—their limited precision in timing market lows underscores the persistent challenges posed by market efficiency. Cryptocurrency markets, characterized by high volatility and noise, exhibit only partial deviations from EMH, making short-term forecasting inherently difficult. Complementing this perspective, behavioral finance highlights how cognitive biases, such as herd behavior, panic selling introduce systematic but noisy patterns [25]. These biases may explain the models' partial success in detecting impending drops, as they capture behaviorally driven volatility.

  However, during extreme sentiment-driven episodes, the models' predictive accuracy diminishes, revealing a critical limitation: even sophisticated algorithms struggle to disentangle signal from noise in markets where investor psychology amplifies unpredictability. Thus, while historical data and behavioral patterns offer some predictive value, the combined effects of market efficiency and irrational investor behavior impose fundamental constraints on reliably forecasting significant cryptocurrency downturns.

5.4 Future Research Directions

Building upon the findings and limitations of this work, several promising avenues for future research emerge:

- **Comprehensive Feature Enrichment:** Systematically incorporate a wider array of features as mentioned in the limitations, particularly on-chain metrics, granular sentiment data, and advanced volatility measures.

- **Advanced Hyperparameter Optimization:** Employ sophisticated hyperparameter tuning techniques for each architecture to potentially unlock better performance.

- **Dynamic Target Variables and Adaptive Models:** Explore methods for dynamically adjusting the prediction horizon or the definition of a "significant drop" based on prevailing market volatility or regime. Investigate adaptive learning models that can adjust to evolving market dynamics.

- **Probabilistic Forecasting and Uncertainty Quantification:** Instead of point predictions for timing, develop models that output probability distributions over the possible number of days to the minimum, or confidence intervals for the binary prediction, providing a measure of uncertainty crucial for risk management.

- **Ensemble Modeling:** Combine predictions from multiple diverse models (e.g., a TCN, an LSTM-Attention hybrid, and a gradient boosting machine on tabular features) to potentially achieve more robust and accurate forecasts.
- **Full-Scale Event-Driven Backtesting:** Integrate the predictive model outputs into a realistic event-driven backtesting framework. This should simulate trading strategies, incorporate transaction costs and slippage, and evaluate risk-adjusted returns (e.g., Sharpe ratio, Sortino ratio, max drawdown) against relevant benchmarks (e.g., buy-and-hold ETH, BTC).
- **Multi-Asset and Cross-Sectional Modeling:** Extend the modeling approach to a portfolio of cryptocurrencies, potentially leveraging cross-asset correlations, or developing models that predict relative asset performance or identify which assets within a universe are most likely to experience significant drops.
- **Explainable AI (XAI) Techniques:** Apply XAI methods (e.g., SHAP, LIME) to understand which features are most influential in the models' predictions, providing insights into market dynamics and increasing trust in the model outputs.

6 Conclusion

This study investigated the application of various deep learning architectures - including an LSTM, a GRU, a Hybrid LSTM-GRU, a TCN, a Mega TCN, and a Hybrid LSTM-GRU with Multi-Head Self-Attention - for the dual task of predicting the occurrence of significant 7-day price lows and forecasting the timing of these lows for Ethereum (ETH-USD). By incorporating a feature set enriched with external market and sentiment indicators, such as Bitcoin closing prices, 10-Year U.S. Treasury yields, S&P 500 index, and the Crypto Fear & Greed Index, we aimed to improve upon predictions based solely on the asset's own price history.

The empirical results from the validation set indicate that the developed models possess a moderate degree of predictive power for identifying periods of heightened downside risk. The Mega TCN model achieved the highest binary classification accuracy of 63% and an ROC AUC of 0.65, demonstrating an ability to outperform a random baseline in anticipating significant drops. This suggests potential utility for these models as components within a risk assessment toolkit for cryptocurrency investors.

However, accurately forecasting the precise timing of these price lows remains a formidable challenge. The Mean Absolute Error for the regression task averaged around 1.5-1.6 days, highlighting the inherent difficulties in precise market timing in such a volatile and complex environment. While some architectures showed marginal improvements over others, no model achieved a level of timing precision that would be unequivocally reliable for standalone short-term trading decisions.

The study underscores that while advanced deep learning models can extract some predictive signal from historical data, the high noise and random nature of cryptocurrency markets impose significant limitations, especially on timing predictions. The limitations identified, particularly concerning the feature space, data granularity, and the need for robust backtesting, pave the way for substantial future research.

In conclusion, while the direct application of the current models for precise, autonomous trading is not recommended, their ability to flag potential significant downturns suggests they could serve as a valuable supplementary tool within a broader, human-in-the-loop risk management framework. Further refinement, feature enrichment, and rigorous backtesting are warranted to fully explore and potentially enhance their practical utility in navigating the turbulent cryptocurrency markets.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. Software available from tensorflow.org.

[2] Issam Akouaouch and Anas Bouayad. A new deep learning approach for predicting high-frequency short-term cryptocurrency price. *Bulletin of Electrical Engineering and Informatics*, 14(1):513–523, Feb 2025.

[3] Khalid K. Al-jabery, Tayo Obafemi-Ajayi, Gayla R. Olbricht, and Donald C. Wunsch II. 4 - selected approaches to supervised learning. In Khalid K. Al-jabery, Tayo Obafemi-Ajayi, Gayla R. Olbricht, and Donald C. Wunsch II, editors, *Computational Learning Approaches to Data Analytics in Biomedical Applications*, pages 101–123. Academic Press, 2020.

[4] Alternative.me. Crypto fear & greed index. `https://alternative.me/crypto/fear-and-greed-index/`, 2024.

[5] Ran Aroussi. yfinance. `https://github.com/ranaroussi/yfinance`, 2017.

[6] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[7] François Chollet et al. Keras. `https://keras.io/`, 2015.

[8] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015. Accessed: 2025-06-09.

[9] Pasquale De Rosa and Valerio Schiavoni. Understanding cryptocoins trends correlations. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 29–36. Springer, 2022.

[10] Kingma Diederik. Adam: A method for stochastic optimization. *(No Title)*, 2014.

[11] Eugene F Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, 1970.

[12] Federal Reserve Bank of St. Louis. 10-year treasury constant maturity rate (dgs10). FRED, Federal Reserve Bank of St. Louis, 2024.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in Python*. Springer, 1 edition, 2023. Referenced chapters include Chapter 2, 3, 5, and 6.

[16] Ibrahem Kandel and Mauro Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT express*, 6(4):312–315, 2020.

[17] Ayush Khaire.

[18] Runze Lin. Analysis on the selection of the appropriate batch size in cnn neural network. In *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*, pages 106–109. IEEE, 2022.

[19] NVIDIA. Long short-term memory (lstm), n.d. Accessed: 2025-06-07.

[20] Abdulhalık Oğuz and Ömer Faruk Ertuğrul. Chapter 1 - introduction to deep learning and diagnosis in medicine. In Kemal Polat and Saban Öztürk, editors, *Diagnostic Biomedical Signal and Image Processing Applications with Deep Learning Methods*, Intelligent Data-Centric Systems, pages 1–40. Academic Press, 2023.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python.

[22] Burak Pirgaip, Burcu Dinçergök, and Şüheda Haşlak. Bitcoin market price analysis and an empirical comparison with main currencies, commodities, securities and altcoins. *Blockchain Economics and Financial Market Innovation: Financial Innovations in the Digital Age*, pages 141–166, 2019.

[23] Pavlo M Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. 2017.

[24] Ramin Ranjbarzadeh, Annalina Caputo, Erfan Babaee Tirkolaee, Saeid Jafarzadeh Ghoushchi, and Malika Bendechache. Brain tumor segmentation of mri images: A comprehensive review on the application of artificial intelligence tools. *Computers in Biology and Medicine*, 152:106405, 2023.

[25] Robert J Shiller. From efficient markets theory to behavioral finance. *Journal of Economic Perspectives*, 17(1):83–104, 2003.

[26] SuperDataScience Team. Recurrent neural networks (rnn) - the vanishing gradient problem, 2018. Accessed: 2025-06-07.

[27] Keras Team. Keras documentation: Adam.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. Accessed: 2025-06-09.

A Appendix: Supplementary Material and Code Repository

This appendix section provides references to supplementary materials that contain further details on the experimental setup, model implementations, and detailed outputs not fully elaborated in the main body of this report.

A.1 Computational Notebook and Code Implementation

The complete Python codebase, including all data preprocessing steps, feature engineering logic, model definitions (using Keras/TensorFlow), training scripts, and generation of results, is available in a Jupyter Notebook hosted on Google Colaboratory. This notebook serves as the primary repository for the implementation details of this study.

The notebook includes:

- Detailed code for data acquisition and cleaning.
- Implementation of all feature engineering processes, including target variable construction and lag generation.
- Definitions of all neural network architectures evaluated (e.g., LSTM, GRU, Hybrid models, TCNs).
- The Keras `model.summary()` output for each architecture, detailing layers and parameter counts.
- Code for model training, validation, and generation of performance metrics and figures presented in this report.
- All necessary library imports and environment setup notes.

A.2 Additional Figures and Detailed Results

While key figures and summary tables are presented in the main body of the report, more granular results, such as:

- Detailed ROC curves for each model on the validation set.
- Confusion matrices for the binary classification task for key models.
- A comprehensive table including all tracked performance metrics for all models.
- Potentially, visualizations of feature importance or attention weights (if applicable and generated).

are also available within the outputs of the aforementioned Google Colaboratory notebook (Section A.1). Readers are encouraged to consult the notebook for these supplementary visualizations and detailed performance breakdowns.