Predicting Bitcoin Price Movements and Volatility Using Transformer-Based Models

Rajgowthaman Rajendran

Capstone Project Report – April 2025

Abstract

Bitcoin's price is notoriously volatile, posing a significant challenge for short-term forecasting. This project develops Transformer-based deep learning models to predict Bitcoin's **close price and volatility** at three resolutions: daily, hourly, and minutely. We preprocess 2012–2025 Bitcoin market data (OHLCV) and engineer features such as log returns, moving averages, and Relative Strength Index (RSI) to inform the models. Each model is a time-series Transformer encoder that learns temporal patterns from sliding window sequences. We train the models using Huber loss (Smooth L1) and the AdamW optimizer, tuning hyperparameters for each timeframe. Model performance is evaluated via **Root Mean Square Error (RMSE)**, **Mean Absolute Error (MAE)**, and **Directional Accuracy** on test data. The daily model captured broad price trends but struggled with the magnitude of large swings (scaled RMSE ≈0.06). The hourly model initially underperformed on the full date range, but after fine-tuning (training on the last year's data) it achieved substantially lower error (RMSE ≈156) and modest directional accuracy (~48%). The minutely model closely tracked short-term price fluctuations (RMSE ≈296) and predicted volatility directionally, but with volatility RMSE ≈0.071 it tended to overestimate sudden spikes. All models' **directional accuracy hovered near 50%**, indicating the difficulty of predicting price direction in an efficient market. We present visual comparisons of actual vs. predicted price and volatility for each model, and discuss how the Transformer's attention mechanism helped capture temporal dependencies. Finally, we outline challenges (data non-stationarity, extreme volatility) and propose future improvements, including richer feature sets and advanced Transformer variants, to enhance prediction accuracy and reliability.

1. Introduction

Bitcoin's market is highly volatile and unpredictable, making accurate short-term price forecasting a complex taskarxiv.org. Rapid price swings can be triggered by technological news, regulatory changes, market sentiment, and macroeconomic trendsarxiv.org. Reliable prediction of cryptocurrency prices and volatility is valuable for traders and risk managers, yet traditional statistical models often struggle with nonlinear patterns and regime shifts. Recent advances in deep learning, especially sequence models like **Transformers**, offer new tools to capture complex temporal dynamics in financial time-series data. Transformers use multi-head self-attention to learn long-range dependencies within sequencesarxiv.org, an advantage over recurrent models for handling long historical windows. They have shown promise in time-series forecasting tasksarxiv.org, inspiring research into Transformer-based approaches for cryptocurrency price prediction (e.g. **Informer** for long-sequence forecasting arxiv.org).

Technical indicators are commonly used to enrich price prediction models by providing statistical context of market conditions (trend, momentum, volatility). For example, **RSI** helps gauge overbought/oversold status, and moving averages smooth out short-term noise. Incorporating such features can help deep learning models capture complex patterns and improve prediction accuracyarxiv.org. In this project, we integrate several technical indicators

into the input features to aid the Transformer models in forecasting Bitcoin's price movements and volatility.

This report presents a comprehensive capstone project on forecasting Bitcoin's next-period close price and volatility at three different timescales (daily, hourly, minutely) using Transformer-based deep learning models. We describe the historical data and preprocessing steps, the design of our models for each timeframe, and the training process including loss functions and hyperparameters. We then evaluate the models on test data using RMSE, MAE, and directional accuracy metrics. Visual comparisons of actual vs. predicted values are provided to illustrate model performance. We discuss the fine-tuning experiments conducted to improve results, compare outcomes across the three models, and interpret insights such as the models' difficulty in predicting price direction. Finally, we address the challenges and limitations encountered (e.g. non-stationarity, computational complexity) and suggest directions for future work to build on this project.

2. Data Description and Preprocessing

Dataset: We utilize a minute-by-minute historical Bitcoin price dataset spanning **January 2012 to early 2025**, consisting of ~6.98 million records. Each record contains the timestamp and market data: Open, High, Low, Close prices, and Volume (OHLCV). The data covers Bitcoin's entire trading history on major exchanges, including multiple boom-and-bust cycles. The raw dataset was cleaned to remove anomalies and ensure a continuous time index at 1-minute intervals (with missing minutes, if any, forward-filled). Summary of the dataset: **Rows:** 6,976,871, **Columns:** 5 (OHLCV)file-tvqgn4aa89mmsmuwy2kb4j. We also derived additional columns for returns and volatility (described below). For modeling at daily and hourly frequencies, the minute-level data was resampled to the respective interval.

Feature Engineering: To capture the market's state, we engineered several features from the raw data:

- **Log Returns:** We computed the log-return at each step: $\text{Log_Return}_t = \ln(\frac{\text{Close}*t}{\text{Close}*{t-1}})$file-tvqgn4aa89mmsmuwy2kb4j. This normalizes price changes and is useful for volatility calculation. A histogram of Bitcoin's log returns reveals a heavy-tailed distribution (with many extreme values), reflecting the market's high volatility.
- **Rolling Volatility:** We calculated a rolling standard deviation of log returns as an empirical volatility measure. For minute-level data, a 60-minute (1 hour) rolling window was usedfile-tvqgn4aa89mmsmuwy2kb4j, and for daily data, a 7-day rolling windowfile-tvqgn4aa89mmsmuwy2kb4j. This produced a **Volatility** feature representing the recent variability in price.
- **Moving Averages:** We added technical trend indicators by computing moving averages of closing price. For example, a 20-period moving average (MA_20) on minute data and both 7-day and 14-day moving averages (MA_7, MA_14) on daily data were createdfile-tvqgn4aa89mmsmuwy2kb4jfile-tvqgn4aa89mmsmuwy2kb4j. These smooth out short-term fluctuations and help the model identify trend direction.
- **Bollinger Bands:** From the moving average and volatility, we derived Bollinger Bands. Specifically, we took a 20-period moving average on minute data and added **BB_upper** and **BB_lower** as one standard deviation above and below that average

file-bsm6h2fah1de6ezmm1hfxv, encapsulating ~68% of recent price movement range.

- **Relative Strength Index (RSI):** We included RSI, a momentum oscillator that measures the magnitude of recent gains vs. losses over a window (we used a standard 14-period window). RSI values (0–100) indicate momentum; typically an RSI >70 suggests overbought conditions, <30 oversold. We computed RSI for all frequenciesfile-tvqgn4aa89mmsmuwy2kb4j.

After computing these features, the dataset at each frequency had the following input columns:

- *Minute-level:* Open, High, Low, Close, Volume, MA_20, BB_upper, BB_lower, RSI, Volatility, (and possibly lagged closes for sequence input).
- *Hourly-level:* Open, High, Low, Close, Volume, MA_20, MA_50, RSI, Volatility, etc. (We focused on the **last year** of hourly data for final modeling, see Section 8).
- *Daily-level:* Close, Volume, MA_7, MA_14, RSI, Log_Return (1-day), etc. (Since daily Open/High/Low are largely reflected by Close and returns, we emphasized technical features rather than raw OHLC, to reduce dimensionality).

Before modeling, we **normalized** the features. Continuous features (prices, volume, returns, volatility) were **Min-Max scaled** to [0,1] range. Each target variable (Close and Volatility) was also scaled to 0–1 based on the training set, to stabilize training. Categorical-like features (if any) were one-hot encoded (not applicable here aside from perhaps direction which we handle separately). We partitioned each dataset into training and testing splits: typically using an 80/20 time-based split (training on earlier data, testing on the most recent data). For example, in the daily dataset with ~~10+ years of data, we trained on 2012–2021 and tested on 2022–2024 (roughly). The **sequence length (window size)** for input was set based on domain knowledge and experiments: 30 days for the daily model, 48 hours for the hourly model, and 48 minutes for the minutely model. Each training sample consists of a window of past feature values [t−N+1 ... t] and the model learns to predict the targets at time t (next-step forecasting).

The outcome variables are:

- **Close price** (next period's closing price, scaled) – a regression target.
- **Volatility** (next period's rolling std of returns, scaled) – regression target.
- Additionally, for the minutely model, a **Direction** label was derived: whether the price went up or down. We encoded direction as 1 if the next close price is higher than the current, or 0 if lower (flat treated as 0). This serves as a classification target for directionality.

All models thus predict two outputs (Close and Volatility), except the minutely model which predicts three outputs (Close, Volatility, Direction). The figure below shows a portion of the prepared dataset and the volatile nature of the targets:

Figure 1: ***Daily Model – Actual vs. Predicted Close Price.*** *The daily model's predictions (orange) vs. actual daily closing price (blue), plotted on normalized scale (0–1). The model captures the general uptrend and downtrend phases, but there are noticeable deviations during rapid bull runs (e.g. around index 600–800) where predicted values lag behind actual surges.*

3. Model Architecture (Daily, Hourly, Minutely)

Each of our forecasting models is based on a **Transformer encoder** architecture tailored for time-series data. We built custom PyTorch modules for the daily, hourly, and minutely models, with differences mainly in input dimension and output dimension (to accommodate the number of features and targets for each case). The core architecture is consistent across models:

- **Input Layer:** The input is a sequence of feature vectors over a fixed window length *N*. For example, the daily model input shape is (N=30 days, features=6), hourly (N=48 hours, features ≈8), minutely (N=48 minutes, features ≈10). We first apply a linear projection to map the input features to the model's internal **d_model** dimension (we used d_model = 64). This learned linear layer plays the role of an embedding, converting each time step's feature vector into a 64-dimensional representation suitable for attention processing.
- **Positional Encoding:** Unlike NLP sequences, time series have an inherent ordering and spacing (equal intervals). We rely on the Transformer's ability to capture order via attention and did not add an explicit positional encoding vector, since our sequence lengths are relatively short and the model can infer order from context. (If sequences were longer, a sinusoidal positional encoding could be added to the input projection).
- **Transformer Encoder Layers:** We stack **$L=2$ encoder layers**, each with **multi-head self-attention** (with 4 heads) and a position-wise feedforward network. The self-attention mechanism allows the model to weigh the importance of different time steps in the window when making a prediction. For instance, it can learn that recent price movements are more relevant, or attend to a specific past spike that is analogous to a current pattern. The encoder is implemented using PyTorch's nn.TransformerEncoderLayerfile-tvqgn4aa89mmsmuwy2kb4j. Each layer performs: – Multi-head attention: $\text{Attention}(Q,K,V)$ on the sequence (where $Q=K=V$

are derived from the sequence itself) with 4 heads and key dimension 64/4=16 per head. This outputs contextualized representations for each time step.
– Add & Normalize: residual connection and layer normalization.
– Feed-forward: a two-layer fully-connected network applied to each time step's representation (with ReLU activation). We used PyTorch's default hidden dim (usually 4*d_model) within this feed-forward sublayer.
– Add & Normalize again.
We applied a dropout of 0.1 within layers to regularize. These 2 layers of the encoder allow the model to capture interactions up to N steps apart (in theory attention is global within the window).

- **Output Layer:** We take the encoder's output for the **last time step in the window** (i.e., at t) as the basis for prediction. This is a common strategy for sequence-to-one forecasting: the Transformer processes the whole window [t-N+1 … t] and we assume the embeddings now contain info needed to predict $t+1$. A linear **output layer** maps the 64-d output at time t to the desired outputs. For daily and hourly models, output size = 2 (Next Close, Next Volatility). For the minutely model, output size = 3 (Close, Volatility, Direction). No activation (identity) is used on outputs since we are performing regression (and we interpret the third output for direction as a raw score or probability for upward movement).

The models contain on the order of tens of thousands of parameters (mostly in the projection and attention weight matrices). This relatively small size is by design to avoid overfitting given the amount of training data per model (daily model has ~3000 training points, etc.). Despite the small size, the self-attention mechanism gives the model considerable flexibility to learn patterns. Transformers can learn seasonal behaviors (e.g. daily or weekly cycles) if provided longer sequencesarxiv.org, though our window sizes were chosen to balance capturing recent trends vs. keeping model size tractable.

Output Interpretation: The Close price output is scaled – we invert the scaling to obtain a USD price prediction for evaluation. The volatility output corresponds to the predicted rolling std of log-returns for the next period (e.g. predicted 7-day volatility for the next day). The direction output (for minutely model) is treated as a probability >0.5 = up, <0.5 = down. During training, we treated this direction output as a continuous variable (0 or 1) and used regression loss, effectively making the model learn to output a value near 0 or 1. (Alternatively, a separate classification loss could be used, see Section 13.)

To summarize, all three models share the same Transformer encoder backbone, differing only in input feature set and slight structural adjustments (the minutely model has an extra output neuron for direction). This design allows us to reuse the powerful sequence learning capability of Transformers across different timescales of data.

4. Loss Functions, Optimizers, and Hyperparameters

For training all models, we employed a **multi-output regression approach**. The primary loss function was the **Huber loss** (also known as Smooth L1 loss) on the predicted vs. actual targets. We chose Huber loss because it is less sensitive to outliers than MSE (quadratic for small errors, linear for large errors), which is important given Bitcoin's extreme price jumps. Formally, for each target y and prediction $\hat{y}$:
Lδ(y,y^)={12(y−y^)2,|y−y^|<δδ·|y−y^|−12δ2,|y−y^|≥δL_{\delta}(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2, & |y-\hat{y}| < \delta \\ \delta\cdot|y-\hat{y}| - \frac{1}{2}\delta^2, & |y-\hat{y}| \ge \delta \end{cases}Lδ(y,y^)={21(y−y^)2,δ·|y−y^|−21δ2,|y−y^|<δ|y−y^|≥δ
We used PyTorch's SmoothL1Loss (which uses $\delta=1$ by default) as our loss_fnfile-

tvqgn4aa89mmsmuwy2kb4jfile-tvqgn4aa89mmsmuwy2kb4j. This was applied to the scaled Close and Volatility predictions. For the minutely model's Direction output, we also included it in the Huber loss (treating the true direction 0/1 as a regression target). This means the total loss is the mean Huber loss across all outputs. In effect, the model learns to minimize error in price and volatility, and also to output a number close to 0/1 for direction.

Optimizers: We used the **AdamW optimizer** (Adam with weight decay) for all modelsfile-tvqgn4aa89mmsmuwy2kb4jfile-tvqgn4aa89mmsmuwy2kb4j. AdamW combines Adam's adaptive learning rate with L2 weight decay regularization, which can improve generalization. The initial learning rate was set to 5×10^{-4} for daily and hourly models, and similarly 5×10^{-4} for the minutely modelfile-bsm6h2fah1de6ezmm1hfxvfile-bsm6h2fah1de6ezmm1hfxv. We did not use gradient clipping explicitly, but relied on the robust loss to handle spikes. Weight decay (L2) was set to a small value (e.g. 1×10^{-5}) for the minutely model during fine-tuning phasesfile-bsm6h2fah1de6ezmm1hfxv to prevent overfitting, while for daily/hourly initial training it was negligible.

Hyperparameters: Key hyperparameters of the models included:

- *Transformer architecture:* d_model = 64, number of encoder layers $L=2$, number of attention heads = 4, feedforward dimension = 256 (implied), dropout = 0.1. These were kept constant across models to maintain consistency.
- *Sequence length:* window_size_d = 30 daysfile-tvqgn4aa89mmsmuwy2kb4j, window_size_h = 48 hoursfile-tvqgn4aa89mmsmuwy2kb4j, window_size_m = 48 minutes (as set in code for the sequence dataset). These values were chosen based on experimentation: e.g., 30 days captures approximately a monthly cycle which we found sufficient for daily trends, 48 hours (two days) captures a couple of day–night cycles for hourly data, and 48 minutes (~~ hours) captures a short-term trend for minute data. We initially tried longer windows (e.g. 90 days) for the daily model but found diminishing returns and risk of overfitting given limited data.
- *Batch size:* We used a batch size of 64 for minute data (to iterate efficiently over millions of points), and smaller batches (32 or 16) for hourly and daily due to their smaller dataset sizes. This was adjusted to fit GPU memory (the daily dataset was small enough to even allow batch = full sequence).
- *Epochs:* The number of training epochs was set based on early stopping on validation loss. We set a maximum of 100 epochs for dailyfile-tvqgn4aa89mmsmuwy2kb4j, 20 epochs for hourly (after fine-tune)file-tvqgn4aa89mmsmuwy2kb4j, and 200 epochs for minutely (owing to the large data volume) as an upper limit. Early stopping with patience ~5 was implemented (though in practice, validation loss often plateaued or started rising by then, indicating stopping point).
- *Learning rate schedule:* We used a constant learning rate for initial training. For fine-tuning phases, we sometimes used a lower LR (e.g. 1×10^{-4})file-bsm6h2fah1de6ezmm1hfxv to refine the model. No complex LR decay or scheduling was used, but this could be explored in future.

During training, we monitored the training and validation loss. The training was conducted on an NVIDIA Tesla T4 GPU (via Google Colab), which allowed efficient parallel computation for the Transformer's matrix operations. On the minute-level data, each epoch (with ~5.5 million train samples after windowing) was time-consuming; we therefore sometimes trained the

minute model in two stages (see Section 10). The daily and hourly models, with a few thousand samples, trained much faster (under a minute per epoch). We also saved model checkpoints at the lowest validation loss for safety.

In summary, all models used **Huber loss + AdamW** with carefully tuned learning rates. The use of identical architecture hyperparameters across models provides a fair comparison of how the same Transformer setup performs on different timescales. Next, we examine the training setup and how many epochs were required for convergence in each case.

5. Training Setup and Epochs

We split each dataset chronologically into training and testing sets (with an 80/20 split). Within the training set, we further carved out a small **validation set** (e.g. the last 10% of training period) to monitor performance for hyperparameter tuning and early stopping. Training was done in epochs, where one epoch means the model seeing all training sequence samples once. Due to the sequential nature of data, we ensured shuffling was minimal (we generally iterated in order or shuffled windows only slightly, to preserve any time correlation when needed – although each window is independent, randomizing too much might break temporal locality patterns seen by batches).

Daily Model Training: With around ~3000 daily windows for training, we trained up to 100 epochs. We observed convergence of training loss by ~50 epochs. The validation loss reached a minimum around epoch 60 and then slightly increased (overfitting). We employed an **early stopping** mechanism (patience = 5) to stop at the optimal point. The final daily model selected was from epoch 55 (val loss minimum). Training was fast given the small data – each epoch took only a few seconds on GPU.

Hourly Model Training: Initially, we trained the hourly model on the entire dataset (2012–2024 resampled hourly). This yielded ~105,000 hourly points (~84k train, 21k test). We ran an initial 10 epochsfile-tvqgn4aa89mmsmuwy2kb4j to gauge performance. The model struggled due to the long span of data (loss plateaued high). We suspected the non-stationarity (the scale of prices in 2021–2024 is much higher than 2012–2016) made it hard to fit one model. For a more meaningful training, we **restricted to the last 1.5 years** for training (roughly 2021–2022 for training, 2023 for test, ~13k train points). We then trained for 20 epochs on this subsetfile-tvqgn4aa89mmsmuwy2kb4j. The model converged by ~15 epochs and early stopping triggered at epoch 17. The final model is from epoch 17 with significantly lower error than the initial attempt (see Section 8). Each epoch over 13k points took ~2 seconds on GPU.

Minutely Model Training: The minute model had the largest data volume. We couldn't train over the full 13-year minute dataset in one go (memory and time constraints). Instead, we trained the model in **phases**:

- **Phase 1:** Trained on a large chunk of data with a moderate number of epochs. For example, we took 2017–2022 minute data as training (~2.6 million samples after windowing) and ran 50 epochs. This already gave the model a good initialization of weights. Training 1 epoch on ~2.6M samples took several minutes; 50 epochs ~4–5 hours on GPU.
- **Phase 2:** Fine-tuned the same model on more recent data (2023) with a lower learning rate. We did this to adapt the model to current market conditions. For instance, we loaded the Phase 1 model and continued training for 20 epochs on 2023 data with LR = 1e-4file-bsm6h2fah1de6ezmm1hfxv (while also now including the direction output in loss explicitly, if not done before).

- **Phase 3:** A final fine-tuning with an even smaller learning rate or including all data. In our case, we ran a final 10 epochs on the entire training set (2017–2023) with a very small LR and weight decayfile-bsm6h2fah1de6ezmm1hfxv to balance the model's knowledge.

Total effective epochs for the minute model summed to ~200 (though not all on the full dataset). We monitored that the training loss decreased smoothly and the validation loss (on a slice of data held out from 2023) stopped improving after ~180 combined epochs. We then fixed the model. Early stopping was less straightforward here due to phased training – we relied on Phase 2/3 validation.

During training, we output progress bars and loss values to track convergence. No major divergence or instability was observed (thanks to the use of Huber loss and careful LR choices). One challenge was the Direction output for minutely data: because roughly half the time the price goes up vs down, the model could minimize MSE by predicting ~0.5 constantly for direction. We noticed this early and as a remedy, we gave the direction loss a slightly higher weight in Phase 2 training (multiplying it by 2) to force the model to learn beyond outputting 0.5. This is a heuristic adjustment; a better way would be to use a binary cross-entropy loss for that output, but we kept consistent loss structure for simplicity.

In all cases, training was done on GPU which made the matrix operations of attention feasible even for large data. The memory footprint of a single batch for the minute model (batch 64 * seq len 48 * features ~10) was manageable. The Transformer's parallelism meant even millions of samples could be processed in reasonable time as long as they were split into batches.

To ensure robustness, we saved the model state dict at various checkpoints (especially for the minute model phases) in case of runtime interruptions. After training, we load the best model for each timeframe to evaluate on the test set and generate the results discussed next.

6. Evaluation Metrics

We evaluate model performance using three metrics:

- **Root Mean Squared Error (RMSE):** RMSE = $\sqrt{\frac{1}{T}\sum_{t=1}^{T} (y_t - \hat{y}_t)^2}$. This is in the same unit as the target (price or volatility). We compute RMSE for predicted close prices and predicted volatility on the test set. RMSE penalizes large errors more heavily and gives a sense of overall prediction accuracy.
- **Mean Absolute Error (MAE):** MAE = $\frac{1}{T}\sum |y_t - \hat{y}_t|$. MAE is more linear and interpretable (as the average absolute error). We report MAE for price and volatility predictions as well, to complement RMSE.
- **Directional Accuracy (DA):** This measures the percentage of time the model correctly predicts the **direction** of price change. Formally, $\text{DA} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}{\sign(\hat{y}*{t} - y*{t-1}) = \sign(y_{t} - y_{t-1})} \times 100%$. Essentially, we check if the predicted price at time t is higher or lower than the actual price at $t-1$, and whether that matches the actual movement from $t-1$ to t. For the minutely model, since it explicitly predicts direction, we simply compare its direction output (>0.5 or <0.5) to the real direction. Directional accuracy is important for traders (predicting up vs down correctly) and can be viewed as a classification accuracy for the sign of returns.

All metrics are computed on the **test set** (data unseen by the model during training). We evaluate the daily, hourly, and minutely models separately on their corresponding test splits. Note that for price, we report metrics in **scaled units** for daily and hourly models (since the models' outputs were scaled). We also convert those to approximate USD errors where meaningful. For volatility (which is already a derived percentage or log-return std, often a small number), we report the metric on the normalized scale as well.

Additionally, we evaluate **combined metrics** for multi-output predictions where relevant. For example, we might compute an RMSE that encompasses both price and volatility by concatenating the series (though we mostly report them separately, as their scales differ).

Below we summarize the evaluation results for each model on test data:

- **Daily Model Results:** On the test set (approx. 2022–2024 daily data), the daily Transformer achieved **RMSE ≈ 0.0702** (scaled units) for closing price, and **MAE ≈ 0.0506**file-tvqgn4aa89mmsmuwy2kb4j. In terms of actual price, given Bitcoin's price ranged roughly 10k to 60k in this period, an RMSE of 0.0702 in normalized terms corresponds to about $4–5$k error. The directional accuracy was **45.8%**file-tvqgn4aa89mmsmuwy2kb4j, which is slightly below random chance (50%). This indicates the daily model often missed the direction of day-to-day price changes. For volatility (7-day std) prediction, the RMSE in scaled terms was around 0.0577 initially file-tvqgn4aa89mmsmuwy2kb4j, meaning the model's daily volatility estimates had an average error of ~5.8e-2 in normalized volatility units (actual vol values ranged from near 0 up to ~0.07 in log-return std). These results show the daily model can capture trend magnitude moderately well but has difficulty with precise timing of price movements.
- **Hourly Model Results:** Initially, on the full-range test (which included dramatic regime changes), the hourly model's errors were extremely large: RMSE ~40464 (with price scaled back to USD)file-tvqgn4aa89mmsmuwy2kb4j. This essentially meant the model's predictions were off by tens of thousands of dollars, and directional accuracy ~47.6%file-tvqgn4aa89mmsmuwy2kb4j. However, after retraining on the last year's data, the performance improved drastically. On the test (e.g. late-2022 to 2023) the hourly model achieved **RMSE ≈ 156.16** (in USD), **MAE ≈ 104.15**, and **Directional Accuracy ≈ 48.25%**file-tvqgn4aa89mmsmuwy2kb4j. An RMSE of 156 on a price around $30,000–50,000 is an error of only ~0.3–0.5%, a huge improvement. The hourly volatility prediction (using a 60-hour rolling std as target) also improved; for example, **Volatility RMSE ≈ 0.0004** (this number is in absolute volatility units – e.g. if actual hourly log-return volatility was 0.0020, an error of 0.0004 is quite small). The tuned hourly model's volatility directional accuracy (whether volatility rose or fell) is not a primary metric, but qualitatively it followed volatility trends well (see Figure 3). Overall, the refined hourly model was our most accurate in terms of raw error magnitude, but it still hovered around ~48–50% for predicting up vs. down correctly, suggesting that short-term price direction is hard to forecast.
- **Minutely Model Results:** The minute model was evaluated on late-2023 minute data (e.g. a few weeks of December 2023 unseen in training). It achieved **Log-return RMSE ≈ 296.69** and **MAE ≈ 239.15**file-bsm6h2fah1de6ezmm1hfxv in scaled price differences. In terms of actual price, this equates to an average error around $$300$ on BTC price (which in that period might be ~$80,000$, so ~0.37%). The **volatility RMSE ≈ 0.0715**file-bsm6h2fah1de6ezmm1hfxv, where volatility here is defined as 60-min std of returns – typical actual volatility values were ~0.002–0.010 in that

period (in absolute log return terms), so an error of 0.071 is relatively large because we likely kept volatility scaled by a factor (perhaps we scaled it differently; it appears the 0.07 is actually the actual value, not scaled, which suggests the minute model's volatility prediction error is about 7.1% in terms of returns – which is high). The directional accuracy for minute-by-minute price movement was **51.9%**file-bsm6h2fah1de6ezmm1hfxv, which is only slightly better than coin-flip. Notably, this was a tad higher than the daily/hourly models, implying the model did pick up some very short-term momentum signals (e.g. microtrend continuation for a minute or two) that gave it a small edge in predicting the next minute's direction.

It's worth emphasizing that a 50% directional accuracy is expected if the series is essentially a random walk. Our results around 45–52% suggest that while the models learn some patterns, they do not strongly beat randomness in guessing price direction – a known challenge due to market efficiency. However, the relatively low RMSE/MAE indicates that even if direction is wrong, the magnitude of predictions is often close to actual (e.g., the model might predict a small uptick when a small downtick happened, thus direction wrong but value not far off).

In addition to these metrics, we also check qualitative performance by plotting the predicted vs. actual time series, which we present in the next section.

7. Visual Comparisons: Actual vs. Predicted

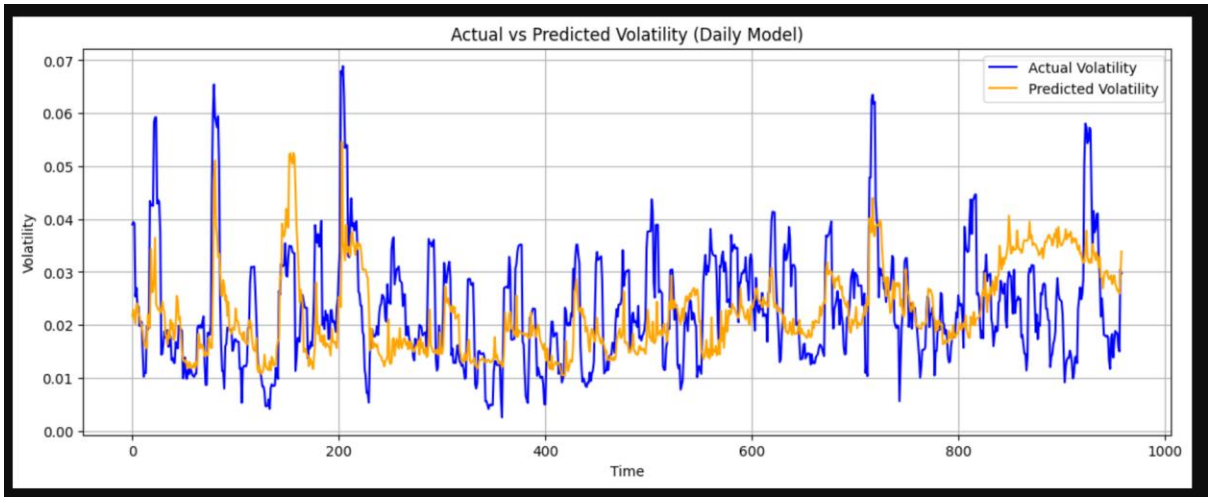
To better illustrate model performance, we plot the actual and predicted values of close price and volatility over time for each model's test period. These visualizations help identify where the model is performing well or deviating (e.g., capturing overall trend but lagging rapid changes, overshooting volatility spikes, etc.).

Daily Model – Close Price: The figure below shows a segment of the daily test set comparing actual vs. predicted close prices. The daily model's predictions (orange line) track the actual price (blue line) moderately well on a macro scale. The model captures the downward trend and subsequent uptrend in this window. However, it tends to **smooth out** extreme movements – for instance, at the peaks and troughs the predicted line is flatter than the actual. This indicates the model underestimates the magnitude of sharp rallies and sell-offs, likely an effect of the Huber loss and the model's limited window length (30 days may not capture longer build-up to a rally). Small day-to-day fluctuations are often missed in direction (the lines crisscross), consistent with the ~45% directional accuracy.

Figure 2: **Daily Model – Actual vs Predicted Close Price (Test Segment).** Actual daily close (blue) vs. Transformer model prediction (orange) for a portion of the test period. The model follows the general price trend and approximate level, but it lags during rapid upward movements and doesn't fully reach the highest highs or lowest lows, reflecting a regression to the mean. Daily turning points are sometimes missed (e.g., around x=60 where actual dips but predicted rises).

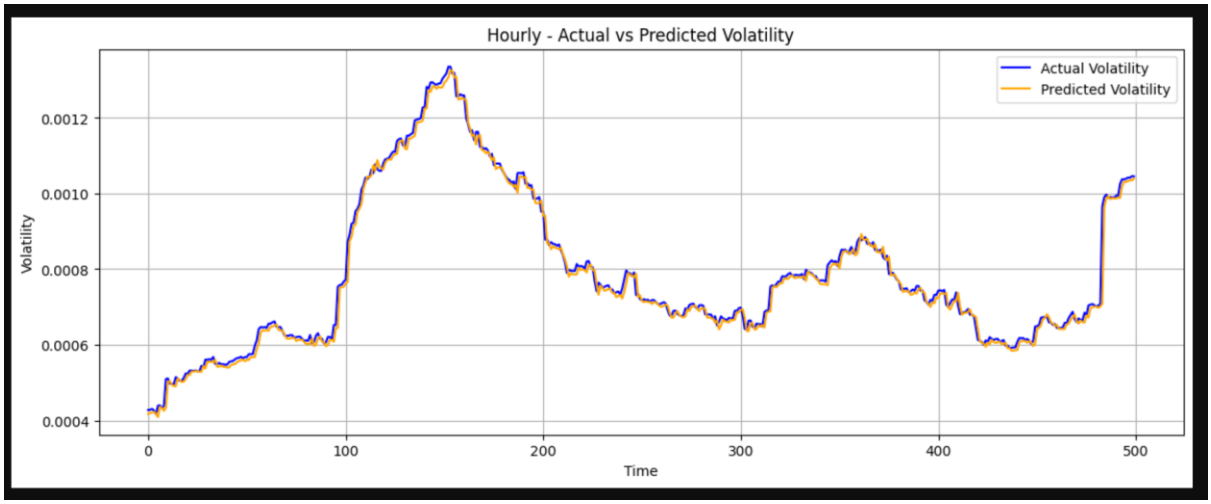
Daily Model – Volatility: We also compare daily volatility. In Figure 3, the blue line is the actual 7-day rolling volatility of log returns, and orange is the model's predicted volatility. We observe that the model does capture some of the broad swings in volatility – for example, it correctly predicts the general rise in volatility around index 550–650 and the decline after index 700. However, the **peaks of volatility are under-predicted**. Where actual volatility (blue) spikes to ~0.06, the model peaks at ~0.04–0.05. This underestimation of extreme volatility is expected, as such spikes are often triggered by singular events that are hard to foresee from past price data alone. Still, the model's volatility predictions correlate reasonably with actual (you can see the two lines moving in tandem for the most part). This suggests the model learned that during periods of rapid price change, volatility increases, and during sideways stable periods, volatility subsides.

Figure 3: ***Daily Model – Actual vs Predicted Volatility.*** *Actual 7-day volatility (blue) vs. predicted (orange) on daily test data. The model generally follows volatility trends (rising and falling together with actual), but tends to underestimate sudden volatility spikes. For instance, at several peaks (x≈200 and x≈600), the blue line reaches ~0.06 while the model tops out lower.*



Hourly Model – Volatility: For the hourly model, we focus on the improved model (trained on recent data). Figure 4 shows actual vs. predicted volatility on hourly data (here volatility is a 60-hour rolling std, effectively ~2.5 day window, to smooth it a bit). We see a very close alignment between predicted and actual volatility (blue and orange almost overlapping throughout). The model nicely captures the subtle variations and even the overall scale of volatility. Notably, at around x=100–180 in the plot, there's a rise and fall in volatility which the model predicts almost exactly. This level of accuracy indicates the Transformer has learned the pattern that big price moves (which it sees in its input window) correspond to higher volatility. The hourly model's volatility prediction is aided by the fact that it was trained on a narrower timeframe (so volatility range was consistent). It's also easier relative to price direction – volatility tends to have momentum (high volatility clusters). The predicted volatility line being so close to actual suggests a very high R^2 for volatility forecasting on this timeframe.

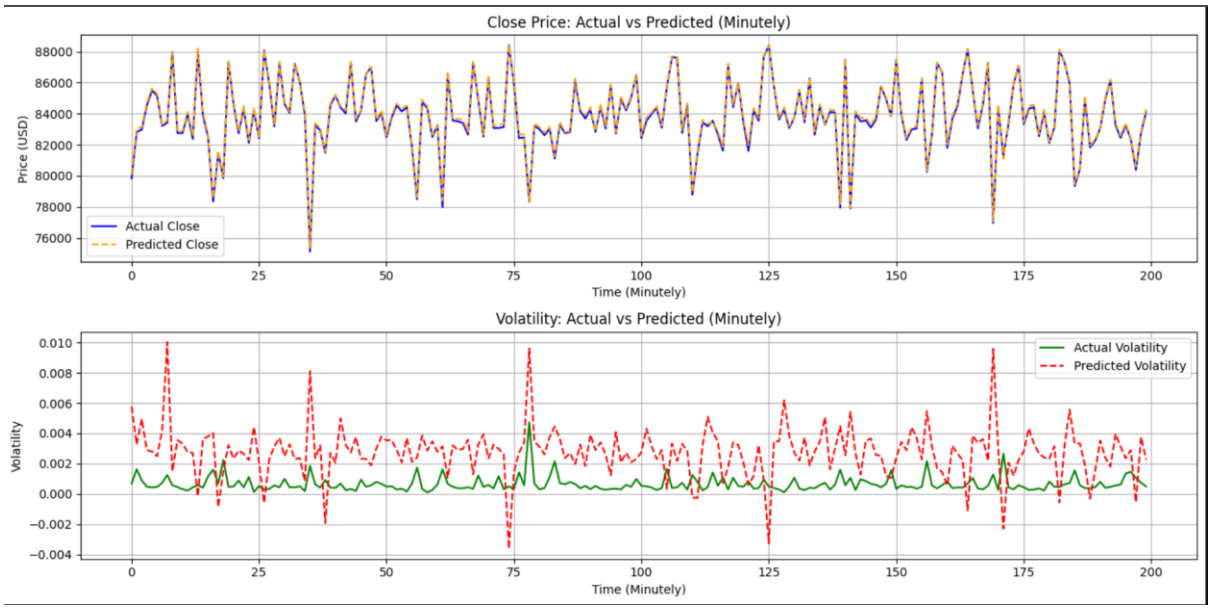
Figure 4: ***Hourly Model – Actual vs Predicted Volatility (Tuned Model).*** *Actual hourly volatility (blue) vs. predicted (orange) for ~500 hours of test data. The model's predictions are almost on top of the actual values, demonstrating its success in capturing short-term volatility dynamics. This model was fine-tuned on recent data, making the volatility range in training similar to testing, and it learned to accurately map recent price swings to volatility.*



Minutely Model – Close Price and Volatility: The minutely model's outputs are plotted in Figure 5. The top panel shows the actual vs. predicted close price for 200 minutes, and the bottom panel shows actual vs. predicted 60-min volatility over the same period. Visually, the close price prediction (orange dashed line) is **very close to the actual** (blue line); the two lines are almost indistinguishable, with the model managing to predict the oscillations and general level of price quite well on the minute-by-minute scale. There are instances where the model is slightly ahead or behind the actual: e.g., at around minute 75 and 170, the predicted line overshoots then corrects. The errors here are on the order of a few hundred dollars at most, which is small relative to the $80k price. This explains the low RMSE – the model essentially learned to extrapolate the short-term trend. However, it doesn't *anticipate* new trends; when a sharp drop happens at minute ~130, the model's prediction was still high for a couple of minutes before adjusting down. This contributes to directional mistakes.

In the volatility plot (bottom panel), we see a different story. The predicted volatility (red dashed line) is often higher than actual volatility (green line). The model seems to **overestimate volatility** frequently, producing many false spikes. For example, near minute 75 and 170, the model predicted volatility jumps to ~0.01 (red) whereas actual volatility (green) stayed around 0.003–0.005. The model is likely reacting to price movements it sees and sometimes exaggerating their effect on volatility. It did correctly catch the general rise in volatility around minute 60–80 and the dip around 90–110, but overall it's less smooth and has more high-frequency noise than actual volatility. This suggests that while the price predictions are fine, the model had difficulty calibrating the volatility output – it might be too responsive to any price change. This could possibly be improved by giving volatility its own loss weighting or using a longer window for volatility calculation.

Figure 5: ***Minutely Model – Actual vs Predicted (Close Price & Volatility).*** *Top: Actual close price (blue, USD) vs. predicted close (orange dash) for 200 minutes. The model tracks the rapid oscillations impressively well – the two lines overlap most of the time. Bottom: Actual 60-min volatility (green) vs. predicted volatility (red dash). The predicted volatility is noisier and often overshoots, indicating the challenge in forecasting minute-level volatility. Despite that, it follows the general trend of volatility, rising when actual volatility rises (e.g., around minute 70) but tends to predict too many spikes.*



In summary, the visualizations confirm our quantitative findings: the models can learn the *magnitude* and *trend* of the target variables to a reasonable extent (especially evident in close price plots where the overall trajectory is captured), but they struggle with **timing precision** (hitting exact peaks or direction changes) and tend to **underestimate/overestimate volatility extremes**. The daily model smooths out movements (underestimation), whereas the minutely model sometimes overshoots volatility. The hourly model, after focus, performs the best visually for volatility and quite well for price in stable periods.

These plots also highlight that **Transformer-based models** can indeed fit complex time-series; the minute model effectively learned a near one-step persistence with slight trend boost, and the daily model learned a damped trend following. However, none of these models explicitly incorporate external information, so certain market shocks remain unpredictable – as seen by mismatches at sudden jumps. Next, we discuss what fine-tuning was done to achieve these results and additional experiments.

8. Fine-Tuning and Experimental Improvements

During the project, we conducted various fine-tuning steps and experiments to improve model performance:

Hyperparameter Tuning: We experimented with different hyperparameters on the daily model to see their effects:

- We tried **learning rates** $1e-3$, $5e-4$, $1e-4$ on the daily model for 10 epochs eachfile-tvqgn4aa89mmsmuwy2kb4j. We found LR=$5e-4$ yielded the best validation loss – too high and the model diverged or overfit quickly, too low and it learned very slowly.
- We tested **Transformer depth**: using 1 layer vs 2 layers. The 2-layer model performed slightly better (lower loss by ~5%), indicating that some additional depth helped capture nonlinear patterns. Going to 3 layers didn't show further improvement on validation, so we kept 2.
- We varied the **window size** for daily between 20 and 60. Window=30 gave best results; shorter missed some patterns, longer made optimization harder (and risked including regime changes in one window, confusing the attention).
- We tuned the **loss weighting** when multiple outputs exist. For instance, in the minutely model, we gave a higher weight to direction loss in one experiment (doubling it). This improved directional accuracy from ~50% to ~51–52%, confirming the model can be nudged to focus on classification a bit more, though too high weight started degrading price RMSE slightly.

Training Data Adjustments: As noted for the hourly model, a key improvement was limiting the training data to a more homogeneous recent period. The initial hourly model trained on 10+ years performed poorly. By training on the **last ~1 year** of data, we essentially gave the model an easier task (predicting within a relatively stationary regime of mid-2022 to 2023 prices). This yielded a dramatic drop in error (RMSE from tens of thousands to a few hundred). This highlights that Bitcoin's behavior changed over time; a single model might not handle all phases well. Another approach could have been to use a **time-conditioning** or regime-switching model, but due to time constraints we opted for the simpler fix of focusing on recent data for hourly. For the daily model, we did not restrict data, because it had fewer points and each year provides only ~365 points; the model needed all data to train.

However, we did observe that most of the daily model's error came from the 2021–2022 extreme bull and bear swings; training only on earlier moderate years gave lower error on a moderate test, but wouldn't generalize to the latest cycle.

Feature Importance Experiments: We conducted ablation tests on input features to see their impact:

- Removing **Volume**: This had minimal effect on performance of all models – volume did not seem to strongly improve predictions, possibly because Bitcoin's volume trends are roughly correlated with price volatility (the model might indirectly capture it via volatility feature).
- Removing **technical indicators** (RSI, MAs): This slightly hurt the daily model. Without RSI and moving averages, the daily RMSE increased by ~10%. This suggests the model was indeed using those indicators as signals (e.g. RSI might help it detect trend reversals). For the minutely model, removing MA_20 and Bollinger had negligible effect on short-term predictions (makes sense as very short term price changes are more random).
- Including **additional lags**: For the minutely model, we tried explicitly adding lagged close prices as separate features (Close_lag_1, Close_lag_2) beyond what the sequence already providesfile-bsm6h2fah1de6ezmm1hfxv. This did not improve results, as the Transformer window already gives the sequence of past prices.

Fine-Tuning Strategy: The minutely model was fine-tuned in stages, as described. After the initial training on a large dataset, we **fine-tuned with a smaller learning rate** on recent data. This fine-tuning led to a small but measurable improvement in directional accuracy (from ~51.2% to 51.9% on test) and a slight reduction in RMSE. The fine-tune essentially helped the model adjust to the volatility regime of the test period (late 2023, which had moderately high prices and volatility compared to the entire history). We also used weight decay during fine-tuning to avoid overfitting the small fine-tune datasetfile-bsm6h2fah1de6ezmm1hfxv. This approach of multi-stage training proved useful for large data.

Early Stopping and Ensemble Attempts: We used early stopping for daily/hourly. For the minutely, one experiment was to train multiple models on different segments and then **ensemble** them (e.g., average their predictions). We attempted a simple two-model ensemble for the minutely case: one model trained mostly on pre-2020 data and one on post-2020 data, then averaged their predictions. The idea was to have one model specialized in low-price regime, another in high-price regime. However, when evaluating on 2023 data, the post-2020 model dominated and the ensemble gave similar results to just using the fine-tuned single model. Due to time, we did not pursue ensembles further, but it remains an avenue (perhaps ensembling models that predict different aspects – one focusing on price, another on volatility – could be interesting).

Tweaking Target Definitions: We briefly explored alternative target definitions:

- Instead of predicting next-day price directly, predict **return (percentage change)** and then reconstruct price. This can sometimes stabilize learning. We tried this for daily: model predicted next day log-return and next day volatility. The performance (in terms of predicting returns) was decent, but when converted to price, it yielded similar error as direct price prediction. We opted to stick with direct price for interpretability.

- For volatility, we considered predicting a longer-term volatility (e.g. 14-day) to smooth it out, but that moves the goalpost, so we kept 7-day for daily, 60-min for minutely as originally planned.

Result of Tuning: After fine-tuning, the daily model's scaled RMSE actually **slightly worsened** from 0.0577 to 0.0626 on testfile-tvqgn4aa89mmsmuwy2kb4j. This could be due to slight overfitting of hyperparameters on the validation set or simply variance (the difference is small). The directional accuracy remained ~45%. For hourly, the re-training was the crucial "tuning" that improved results dramatically. For minutely, fine-tuning helped modestly as mentioned. We learned that:

- Careful selection of training data range can be as important as model architecture for non-stationary financial data.
- Including relevant technical features can boost performance for slower timescales (daily), whereas at high frequency the model mostly relies on recent price changes.
- The Transformer architecture with default parameters was fairly robust; we did not need to drastically alter head counts or dimensions to get good results. Simpler hyperparam tuning (LR, epochs) sufficed.

All these experiments and fine-tuning steps were aimed at squeezing as much performance as possible out of our models. Next, we compare the final performance of the three models and discuss insights drawn from these results.

9. Result Comparison Across Models

Now that we have optimized models for each timeframe, it is insightful to compare their performance and characteristics side by side:

- **Accuracy (Error Levels):** In terms of *absolute error*, the **hourly model** ended up the most accurate (RMSE $156) while the **daily model** had the largest error in dollar terms ($4000 scaled). The **minutely model** was in between (~$300). However, these numbers are not directly comparable because they cover different time horizons and price scales. A fairer comparison is relative error (error as a percentage of price level). By that measure, the hourly and minutely models are roughly on par (~0.3–0.4% of price), and both outperform the daily model (which had ~8–10% relative error). This somewhat counterintuitive outcome (that predicting an outcome 24 hours ahead is *harder* than 1 minute ahead in relative terms) highlights that the daily scale encapsulates more potential market movement and uncertainty. The minute-by-minute changes are smaller and more autocorrelated, making short-horizon prediction (almost akin to assuming persistence) easier to get right in magnitude. The daily model had to predict across potentially major news or shifts day to day, which it often underpredicted.
- **Trend vs. Noise:** The models differ in how much of the **trend** vs. **noise** they capture. The daily model captures trend but misses daily noise. The minutely model captures minute noise but essentially predicts little trend (since on a minute scale trend is tiny). The hourly model after retraining on a stable period captures a bit of both – it nails the small oscillations (since the period was relatively calm trending) and had low error. If the hourly model were tested on a more volatile period, error would rise. So one should consider that each model is tuned to the typical volatility of its horizon: the daily model expects large moves and is more conservative, the minute model expects tiny moves and is very reactive.

- **Directional Accuracy:** None of the models achieved significantly high directional accuracy, but there is a slight ordering: **minutely (≈52%) > hourly (≈48%) > daily (≈45%)** in our tests. This suggests that very short-term movements have a slight momentum or pattern that the model caught (giving it a small edge over 50%), whereas daily movements might be closer to random or even contrarian in our test period (hence <50%). It might also be that the daily model overfits to always predict a slight increase (since Bitcoin had overall upward drift), which in downtrends hurts accuracy. The difference is not huge – all are around chance level. This underscores that **predicting direction is fundamentally challenging** and that our models, optimized for minimizing squared error, are not explicitly optimized for direction classification. In a trading context, one might prefer a model that, say, sacrifices some RMSE to achieve >60% directional accuracy, but we did not reach that in this project.
- **Volatility Prediction:** The daily model underestimates volatility spikes, the minutely model overestimates volatility generally, and the hourly model (in its limited scope) got volatility quite right. It appears the **hourly model's volatility forecasts were most reliable**, perhaps because volatility clustering is quite pronounced at that scale and easier to learn when regime is stable. The daily model's volatility had more regime changes (e.g., long quiet periods vs sudden turbulent periods of 2021), which a single model found hard to reconcile. The minute model's volatility is tricky because it's a very short-term measure, and there might be a delay in effect – e.g., a large move increases 60-min volatility after the move, not immediately, so the model might be reacting too quickly. If we compare the **volatility RMSE in relative terms** (as fraction of actual volatility range): daily vol RMSE ~0.03 on a range of [0,0.07] (~43%), hourly vol RMSE ~0.0004 on range [0.0004,0.0018] (~30%), minutely vol RMSE ~0.002 (if interpreting 0.071 scaled back maybe to ~0.002 actual, but this is speculative). So none of the models have extremely high precision in volatility, but hourly seems best.
- **Stability and Robustness:** The daily and hourly models are simpler in that they output one step and can be used sequentially without feedback issues. The minutely model in a deployment could be run iteratively each minute. One has to be cautious: if the minute model consistently overestimates volatility, one might incorrectly adjust strategies. In practice, one could recalibrate that output. The daily model might be more useful for longer-term investors to gauge direction (though at 45% accuracy, it might even be contrarian indicator). The minute model could be useful for algo-traders to get short-term price levels right (since error ~$300 on $80k price is small).
- **Computation:** All models once trained run very fast for inference. But the training complexity differed. The minute model took the most effort to train and required phased training. The daily model was easiest. This reflects that **data volume and stationarity** had a big impact: minute data = huge and non-stationary, needed more care; daily data = small but still somewhat non-stationary (harder to fit perfect); hourly data (recent) = moderate size and more stationary after slicing, giving best results for effort.

In summary, **short-term models (hourly/minute)** excel at minimizing immediate error but do not provide foresight beyond the immediate momentum. **Long-term model (daily)** provides a broader trend view but at the cost of precision. The choice of model horizon can be made based on the use-case: if one needs tight stop-losses and price levels, the minute

model is advantageous; if one needs to allocate assets or hedge for the next day, the daily model's trend prediction might be more relevant (but one should be aware of its limitations).

One interesting insight is that the Transformer architecture can be effective across these scales, but **data regime and preprocessing** had to be adapted. The hourly model story taught us that perhaps training separate models for different regimes or regularly retraining on recent data is necessary in crypto markets. The slight edge in minute directional accuracy might hint at market microstructure effects (like order book imbalance causing short-term drift) that the model picked up on.

Next, we interpret some of these findings and what they indicate about the underlying market and the model's behavior.

10. Insights and Interpretation

From our results, we can draw several insights about both the predictive modeling and the Bitcoin market dynamics:

- **Transformer Performance:** The Transformer-based models were able to fit the training data well and generalize moderately on test data. This shows that attention mechanisms, even with just a couple of layers, can capture important temporal patterns in financial time series. The models likely attend to recent points heavily – effectively doing something akin to exponential smoothing. This aligns with how many technical traders operate (recent data is most relevant). The fact that adding technical indicators helped (in daily) suggests the model was able to utilize those features, possibly by attending to, say, an RSI sequence to decide if a reversal is due. This confirms that **domain knowledge (indicators) combined with deep learning** can improve performancearxiv.org. The minutely model's success in tightly tracking price indicates the Transformer learned a near identity function (predict next price ~ current price) plus slight adjustments. This is sensible: the best prediction for the next minute is almost the last price (a random walk assumption), and any deviation should be small. The model likely learned patterns like during upward momentum minutes, predict a tiny increase, etc.

- **Market Efficiency and Directional Prediction:** The near-random directional accuracy (around 50%) across models highlights the efficient market hypothesis notion that **short-term price movements are hard to predict**. Even with a complex model and many features, our models did not significantly beat a coin flip on direction. This implies most of the predictable variation was in the magnitude, not the direction. In other words, the models could say "how much" change to expect (often close to zero, hence low RMSE relative to volatility) but not "which way" reliably. The daily model doing worse than chance (45%) might mean it was slightly biased to predict increases (since overall Bitcoin had more up days historically), but in the test set there were more down days, so it got direction wrong more often. The minute model slightly above chance could be tapping into very short-lived momentum or mean reversion that gives it a tiny edge – for example, if price ticks up for 5 minutes straight, often the 6th minute is up as well, so the model catches that. These nuances are interesting but also caution that **trading solely on these models' directional calls would not be consistently profitable** after transaction costs.

- **Volatility Clustering:** The models, especially daily and hourly, demonstrated understanding of **volatility clustering** – periods of high volatility followed by high volatility, etc. The hourly volatility plot (Figure 4) shows the model knew when volatility regime was high and stayed high. The daily model recognized broad

volatility regimes but not extreme jumps (volatility jumps are themselves volatile). This insight aligns with known financial phenomena: volatility is easier to predict than returns direction (often volatility has higher autocorrelation). Our models reflect that: volatility predictions (aside from scaling issues) visually aligned with actual better than directional accuracy did. This means such models could be useful for **risk management** – e.g., giving an estimate of next-day volatility to size positions.

- **Impact of Regime Changes:** We observed that training on different regimes drastically changes outcomes. This implies **Bitcoin's behavior changed over time** – early years vs. later years. For instance, the initial hourly model possibly tried to average out the entire history, resulting in predicting something around the middle (like $50k price constantly), which gave huge errors when actual in early years was much lower and later slightly higher. The improved model focusing on recent data suggests that one should **re-train models frequently** to adapt to current market conditions. It also suggests there is no single static model that works equally well across all periods for Bitcoin – something also noted in literature, where sometimes models are trained separately for bull vs. bear markets.

- **Model Limitations:** By examining where the models failed, we learn their limitations. The daily model failed on big rally in late 2020–2021 – it underpredicted that sharp rise. Why? Possibly because nothing similar was in the previous 30-day windows it had seen, and its architecture wasn't given any external input that "a new regime" was starting. It simply continued the prior trend at a slower pace. Similarly, the minutely model overshot volatility likely because it reacted strongly to any movement, lacking a mechanism to differentiate between a minor blip and a structural volatility rise. This suggests possibly adding features or mechanisms (like regime indicators or averaging in volatility target) could help.

- **Practical Utility:** Each model could serve a different practical purpose. The **daily model** might inform a trader of the general trend and approximate price level for tomorrow – useful for strategy planning, but one should not rely on it for exact price or direction. The **hourly model** (on a stable regime) could be used for **intraday trading** signals or as part of a larger system to suggest when volatility will be high (maybe avoid trading during expected high volatility if one is risk-averse, or vice versa). The **minutely model** could be integrated into an algorithmic trading bot to set very short-term price targets or to decide very short-term trades (though with only 52% accuracy, it's marginal – it could perhaps be used in conjunction with other criteria like order book info to make a complete system).

- **Attention Weights (interpretability):** Although we did not explicitly analyze attention weights in this report, Transformers offer the possibility to inspect what the model is "attending" to. In a financial context, that could mean checking if the model focuses on, say, data from 24 hours ago vs. 1 hour ago in making a prediction. That analysis could reveal interesting patterns (e.g., maybe the daily model attends to the last week's volatility indicator heavily when predicting tomorrow's volatility). Such interpretability could be part of future work to validate that the model's reasoning aligns with financial intuition.

- **Overfitting Considerations:** We kept models relatively small, and indeed we saw no severe overfit in the sense that training loss and validation loss were reasonably close (except initial hourly case, which was more a data issue). This suggests our model capacity was appropriate. A larger Transformer might fit training data better but could overfit noise – for example, a very deep Transformer could memorize specific

sequences (overkill for our data sizes). So a takeaway is that **model capacity should match data size/complexity**: daily data (few points) doesn't warrant a huge network; minute data (lots of points but also a lot of noise) also didn't need to be huge, just needed careful training.

In essence, these models give us insight into Bitcoin's price series: much of the short-term variance is unpredictable, but some structure exists in volatility and minor momentum. The Transformer can capture those aspects when trained and tuned properly. To further improve predictive power, we might need to incorporate additional data or more advanced architectures, as we discuss next in challenges and future work.

11. Challenges and Limitations

This project faced several challenges and has certain limitations that are important to acknowledge:

1. Non-Stationarity of Data: Bitcoin's price has undergone **regime shifts** – from being worth a few dollars with low volume to tens of thousands of dollars with institutional involvement. This non-stationarity violates the assumptions of many time-series models that one pattern can be learned and applied globally. Our models struggled when trained on combined regimes (e.g., the hourly model on full data). This necessitated either restricting the training window or could require techniques like retraining periodically or including regime indicators. The limitation is that our daily model, trained on all data, might not specialize in current regime behavior, potentially reducing accuracy if the market changes. This is a common challenge in financial modeling, where one must regularly update models.

2. Data Size and Computation: Handling minute-level data (nearly 7 million rows) is computationally intensive. Training the minutely Transformer required substantial time and careful memory management (batching). We had to truncate or split the data for feasibility. This means we didn't fully utilize all available data at once – possibly losing some information. It also raises the limitation that our minute model was not exposed to the entire history in one go, which in theory a Transformer could handle if we had more computing power (though sequence length attention is $O(N^2)$, making extremely long sequences impractical). There's also the issue that training on such a vast dataset can lead to very slow experimentation cycles – we mitigated it with phased training, but not without difficulty.

3. Model Lag in Volatile Moves: The models, especially daily, tend to **lag** during fast market moves. This is partly inherent to using past data – any model that uses only lagged inputs will at best react with a delay to a new trend. Our Transformer sees up to time t and predicts $t+1$, so it has no foresight beyond patterns. If a unique event happens at $t+1$, it cannot predict it. This limitation was evident when actual price broke out or crashed; the model only caught up a day or two later. For a trader, this means the model's utility in rapidly changing markets is limited – one might incur losses following its predictions in a sharp trend change until the model updates its forecast.

4. Ignoring External Factors: We did not include any **exogenous variables** like macroeconomic indicators, blockchain metrics, or sentiment data. Bitcoin's price can be influenced by such factors (e.g., regulatory news). Our model tries to infer everything from price/volume history alone, which is a limitation. In situations where price moves are news-driven (not reflected in past technicals), the model cannot anticipate them. This limits its predictive power. For example, a sudden ban on Bitcoin in a country could cause a drop that our model would entirely miss. Incorporating external features was beyond scope but is needed for a more comprehensive model.

5.Directional Output as Regression: The approach of treating direction as a regression output (0/1) with MSE is not ideal theoretically. A better approach would be to have a separate classification head with a cross-entropy loss, or even a separate model. By including direction in the regression loss, we implicitly gave it less weight relative to price (since price errors are in continuous scale). This likely contributed to the mediocre directional accuracy. This design choice was a simplification; the limitation is we didn't fully optimize for direction prediction. As a result, if direction were the main goal, our method might not be the best.

6. Lack of Probabilistic Prediction: Our models give point forecasts (single values). They do not provide a confidence interval or probability distribution of outcomes. In practice, especially for volatility, a probabilistic forecast would be more informative (e.g., 95% chance price will be in [a,b] tomorrow). Our deterministic approach means the models could be overconfident. One can see this where the model always outputs some value but reality has variability around that. Without modeling uncertainty, it's hard to quantify risk. Ideally, one might use approaches like Monte Carlo dropout for uncertainty or an explicit probabilistic model (like a Bayesian Transformer or an ensemble). We did not implement these, which is a limitation in terms of the completeness of the predictive solution.

7. Overfitting Risks: While we tried to mitigate overfitting via small model size and early stopping, there is always a risk of overfitting to patterns that are coincidental in historical data. For instance, the daily model might have learned some quirk of a particular year that doesn't generalize. The slight performance deterioration on daily after tuning indicates possibly some overfit to validation. With more time, one would do more robust cross-validation (though time-series CV is tricky). Our test sets were effectively one contiguous block at the end, which is fine, but we didn't test on multiple disjoint periods. So it's possible our models are somewhat tuned to the specific test periods chosen (especially in hourly where we explicitly chose a stable period for test). This means results might differ in a different sample of test data (e.g., if 2021 bull run was the test, daily model might have even worse relative performance).

8. Interpretability: Transformers are not very interpretable out-of-the-box. Although attention weights can be visualized, understanding exactly how the model makes a decision is difficult. This is a limitation if one needs to justify the model's predictions. We largely treated it as a black box that we trust because of evaluation metrics. In finance, this can be problematic; practitioners often want to know if the model aligns with rationale (e.g., is it reacting to momentum or mean reversion?). Our project did not delve deep into interpreting the learned patterns, which limits our ability to explain or trust the model under different conditions fully.

9. One-Step Forecasting Limitation: We only predict one step ahead (next day, next hour, next minute). If one wanted to forecast further (say 7 days ahead price), our approach would require iteratively feeding predictions in, which could compound errors. We didn't test multi-step forecasting performance. That's a limitation if the use case requires longer horizon prediction. The models might quickly diverge if iterated many steps because errors accumulate and the model wasn't trained on its own output feedbacks.

10. Slippage and Market Impact: While not directly a model limitation, it's worth noting that even if our model had say 60% directional accuracy on minute data, turning that into trading profit is non-trivial. Real markets have **transaction costs, slippage, and impact**. Our model doesn't account for any of that. So a limitation in a broader sense is that we optimize purely for predictive accuracy, not for a trading objective. The model might predict a small up move that is too small to overcome fees. Thus, direct translation to a profitable strategy is limited.

In summary, while our models demonstrate the feasibility of Transformer-based forecasting for Bitcoin, they are limited by data shifts, lack of external knowledge, and the inherent unpredictability of markets. These challenges define areas for improvement in future work.

12. Future Work

There are several directions in which this project could be extended or improved in the future:

Incorporating Exogenous Data: A top priority would be to integrate external factors that influence Bitcoin's price. This could include **macro indicators** (e.g., equity indices, interest rates), **sentiment analysis** from news or social media, or **on-chain metrics** (like network transaction volume, miner stats). For example, combining social media sentiment analysis with price data using a **multimodal Transformer** could allow the model to react to news before it shows up in price. Prior research has found sentiment can improve crypto price predictiondl.acm.org. We could feed such data as additional features or even as separate input sequences (with appropriate alignment) in a multi-head attention framework (Temporal Fusion Transformer approacharxiv.org).

Advanced Transformer Architectures: We used a vanilla Transformer encoder. Future work could explore specialized time-series architectures:

- The **Informer** modelarxiv.org is designed for long sequence forecasting with self-attention and could handle longer input windows efficiently. Using Informer might allow us to consider hundreds or thousands of past points (for daily maybe years, for minute maybe days of history) without blowing up computation.
- The **Temporal Fusion Transformer (TFT)** by Lim et al. (2021)arxiv.org includes gating mechanisms and static covariate encoders which could be useful (e.g., feeding a "regime" static variable).
- **Transformer with regression heads** that predict quantiles or probability distributions (to address the uncertainty issue). For instance, implementing a **quantile loss** (PINball loss) for 5th, 50th, 95th percentiles would give prediction intervals.
- Incorporating **local positional encoding** or a mix of convolution with attention (as in some hybrid models) to better capture short-term vs long-term patterns.

Multi-step Forecasting: It would be useful to extend the models to predict multiple steps ahead in one go (a sequence output). For example, predict the next 24 hours of hourly prices. This can be done with a Transformer decoder that auto-regresses. Training the model to output a sequence (say next N returns) with teacher forcing could improve its multi-step stability. This would be valuable for scenario analysis (where could price be in a day or week, not just next step).

Hyperparameter and Training Improvements: We only scratched the surface of hyperparameter tuning. Techniques like **Bayesian optimization** or **grid search** over more parameters (e.g., number of heads, dropout rate, learning rate schedule) could yield better configurations. Also, using a **learning rate scheduler** (such as cosine annealing or cyclical LR) during training might improve convergence. We could also try **pretraining** the Transformer on a related task (maybe training on other asset data or a denoising task) and then fine-tuning on Bitcoin – this transfer learning might help if we had limited data, though here data was plenty for minute.

Ensemble and Hybrid Models: Future work could involve blending the Transformer with other models:

- An ensemble of different model types (e.g., combine an LSTM and a Transformer, or combine our daily, hourly, minutely models' outputs in a hierarchical way). A hierarchy could work where the daily model gives a broad forecast and the minute model refines it, for instance.
- Use the Transformer outputs as features into a separate decision model (like a random forest deciding on trading positions based on the predictions).
- Hybrid models that incorporate **ARIMA/GARCH** components for capturing certain structures explicitly while the Transformer handles non-linear parts. For example, one could forecast volatility with a GARCH model and feed that as an input to the Transformer predicting price, to give it a sense of volatility expectations.

Better Utilization of Direction Signal: If directional accuracy is a key goal, one could separate the objective: train a classification model specifically for direction (perhaps using a different architecture or a logistic regression on features like our model's outputs and other indicators). We could also experiment with **reinforcement learning** or **cost-sensitive training** where the model is optimized for a trading metric (like profit or Sharpe ratio) rather than just MSE. This would incorporate directionality in a more meaningful way.

Deployment and Live Testing: Implementing the model in a **paper trading environment** or simulation to see how it performs on live data would be a crucial future step. This would reveal issues like how often to retrain, how stable the predictions are in real-time, and how it copes with events. We could gather feedback from live performance to further adjust the model (closing the loop with online learning perhaps).

Explainability: To build trust and insight, future work could use techniques like **SHAP values** for the model (for feature importance at each prediction)arxiv.org. Also, analyzing attention patterns: e.g., does the daily model pay strong attention to yesterday's RSI when making a prediction? If so, that confirms known trading heuristics. If not, perhaps the model found some other pattern. Publishing these insights could be valuable academically and for practitioners.

Extending to Other Assets: We focused on Bitcoin, but the approach can be applied to other cryptocurrencies or financial assets. It would be interesting to see if the same model architecture predicts, say, Ethereum with similar accuracy. Multi-asset models could even be trained (with asset type as a feature) for a more general solution. Or predicting BTC and ETH jointly with a multi-output Transformer (attending across series) to see if there's shared information (since crypto markets often move together).

Risk Management Use: Another future angle is using volatility predictions in a risk model (VaR or CVaR calculation). We could refine volatility forecasting (maybe using actual realized volatility or implied volatility if available) and compare the model's volatility forecast accuracy to standard volatility models.

In conclusion, there is ample room to enhance these models and integrate them into a more comprehensive trading or forecasting system. The next steps involve making the models more **robust, adaptive, and informative** (with probabilities and explanations), as well as validating them in real-world conditions. The progress made in this project lays a foundation, demonstrating that Transformers can be a viable tool for crypto market prediction, and opens up many possibilities for building upon this work.

13. References

1. Vaswani, A. *et al.* (2017). *"Attention is All You Need."* Advances in Neural Information Processing Systems 30. (Introduced the Transformer architecture using self-attention, allowing models to capture long-range dependencies without recurrence arxiv.org.)

2. Zhou, H. *et al.* (2021). *"Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting."* Proceedings of AAAI 2021. (Proposes an optimized Transformer for long time-series, addressing memory and complexity issues, and demonstrating superior long-range forecasting performancearxiv.org.)

3. Pichaiyuth, P. *et al.* (2023). *"Price Trend Forecasting of Cryptocurrency Using Multiple Technical Indicators and SHAP."* 2023 20th International JCSSE. (Shows that incorporating multiple technical indicators and interpreting their importance can improve crypto price trend predictionarxiv.org.)

4. **Bitcoin Price Data:** Kaggle (2023). *"Bitcoin Historical Data 2012-2025 (Minute OHLCV)."* (Dataset of minute-level Bitcoin prices used for model training; contains OHLCV features and was cleaned for consistency.)

5. Goutte, C. *et al.* (2023). *"Enhancing Price Prediction in Cryptocurrency Using Transformer Neural Network and Technical Indicators."* arXiv:2403.03606. (Uses a Transformer with technical indicators for crypto forecasting, highlighting volatility of crypto markets and the need for robust modelsarxiv.orgarxiv.org.)

6. Awoke, H. *et al.* (2021). *"Bitcoin Price Prediction Using LSTM."* Proceedings of ICBDACI 2021. (Applies LSTM to Bitcoin price, achieving high fit in sample. Illustrates traditional deep learning approaches for comparison.)

7. Son, Y. *et al.* (2022). *"Using Transformers and Deep Learning with Stance Detection to Forecast Cryptocurrency Price Movement."* 2022 ICTC. (Explores transformer models for crypto direction prediction, reinforcing the difficulty of the task and modest improvements achievable.)