# SEE LAST 3 PAGES

# High-Frequency Return Forecasting using Transformer Architectures

## 1. Introduction and Problem Statement

In the context of algorithmic trading and quantitative finance, accurately predicting short-term returns can unlock alpha opportunities in highly efficient markets. This project applies machine learning techniques—specifically an encoder-only Transformer architecture—to high-frequency trading data to forecast short-term returns of volatile stocks. The final goal is to develop a return prediction model that maximizes profitability by minimizing directional error and return deviation, under realistic market conditions.

We aim to answer the following question:  
**Can an encoder-only Transformer model capture complex dependencies across time and assets in high-frequency data to produce economically valuable predictions?**

## 2. Data Preprocessing and Feature Engineering

### 2.1 Raw Data Description

The dataset spans from **December 1st to December 31st**, comprising high-frequency price data recorded at regular intraday intervals of 10 minutes. The core variables include:

* DATE: Date in format YYYYMMDD.
* SYMBOL: Stock symbol to which the other data are associated.
* TIME: Intraday time of the data observation in format HH:MM:SS.
* MID OPEN: Mid-price (average of the best bid and best ask) at TIME
* ALL\_EX : exchange activity indicator

### 2.2 Data Cleaning and Filtering

* Selected only stocks with full observations over the period.
* Removed stocks with inconsistent or low liquidity profiles.
* Filtered top N most volatile stocks using standard deviation-based quantiles (this is optional)
* Generate a timestamp by combining TIME and DATE

### 2.3 Target Variable

* **Return**: computed as the **percentage change** in MID OPEN between consecutive timestamps. We renamed it **Target\_Return** after the data cleaning and filtering.
* Normalized across assets and time windows. The values of mean and standard deviation are kept to denormalize after the experiment.

### 2.4 Feature Engineering

* Financial features :
  + Momentum
  + Realized volatility
  + Skewness
  + Kurtosis
  + Rolling mean and std (windows 1h, and 30 min)
* Non-linear features
  + Interactions, polynomial transformations
* Categorical/time embeddings:
  + Day of the week
  + Hour, Minute (cyclic)
  + Symbol ID

## 3. Predictive Model: Transformer Architecture

### 3.1 Model Motivation

Transformers are particularly suited to modeling dependencies over time and across multiple entities (e.g., stocks) without assuming fixed lag structures. We design a custom **encoder-only transformer** to leverage these capabilities for return prediction.

### 3.2 Input Representation

**Input shape: [B, T, S, d]**

* + B: Batch size
  + T: Sequence length (lags)
  + S: Number of stocks
  + d: Feature dimension

### 3.3 Embedding and Projection

* **HFEmbedding:**
  + **Categorical embedding: symbol, day, dayname**
  + **Cyclical embedding: hour, minute**
* **HFINputProjector: Projects features into a common d\_model = 128 space.**
* **Positional Embedding: Encodes relative/absolute time information within each sequence.**

### 3.4 Attention Modes

**Two attention mechanisms were tested:**

* **Across-time attention (per stock): shape [B \* S, T, D]**
* **Across-stock attention (per time): shape [B \* T, S, D]**

### 3.5 Transformer Configuration

**python**

**CopierModifier**

**{**

**"d\_model": 128,**

**"num\_layers": 8,**

**"n\_heads": 16,**

**"dropout": 0.05,**

**"expansion\_factor": 2,**

**"output\_dim": 1,**

**"initial\_attention": "time", # or "cross"**

**"loss\_method": "custom"**

**}**

**Each TransformerBlock includes:**

* **Multihead self-attention with layer normalization**
* **Feedforward GELU activation**
* **Second layer normalization**

**Optional final wrapper layer used for additional smoothing or aggregation.**

### 3.6 Prediction Head

**Outputs a vector of predicted returns per stock and time window. Used for both regression and directional accuracy evaluation.**

## 4. Evaluation Methodology and Performance Metrics

### 4.1 Train-Test Split

* **Time-based split: early December used for training; later dates for testing.**
* **Cross-validation avoided due to time series leakage risks.**

### 4.2 Custom Loss Function

**Our custom loss penalizes both:**

* **Return deviation (mean squared error on continuous return value)**
* **Sign mismatch (directional error)**

**Loss=α∗MSE(pred,target)+(1−α)∗SignLoss(pred,target)Loss = α \* MSE(pred, target) + (1 - α) \* SignLoss(pred, target) Loss=α∗MSE(pred,target)+(1−α)∗SignLoss(pred,target)**

**Where SignLoss penalizes incorrect direction with higher weight.**

### 4.3 Metrics Used

* **Mean Squared Error (MSE)**
* **Directional Accuracy (% of correctly predicted signs)**
* **Sharpe Ratio and Cumulative PnL of a simulated trading strategy**
* **Transaction-cost adjusted returns**

## 5. Results and Analysis

### 5.1 Quantitative Results

| **Model Variant** | **MSE** | **Sign Accuracy** | **Sharpe Ratio** | **PnL (after costs)** |
| --- | --- | --- | --- | --- |
| **Transformer (time)** | **0.000xx** | **xx%** | **x.xx** | **xx.xx%** |
| **Transformer (cross-stock)** | **0.000xx** | **xx%** | **x.xx** | **xx.xx%** |
| **Linear baseline** | **0.000xx** | **xx%** | **x.xx** | **xx.xx%** |

### 5.2 Qualitative Observations

* **Attention patterns show some degree of temporal autocorrelation being learned.**
* **Across-stock attention improves performance when correlation between stocks is high.**
* **Sign accuracy is significantly better than chance, even for volatile assets.**

### 5.3 Figures

* **Cumulative PnL over time**
* **Heatmaps of attention scores**
* **Sharpe ratio comparison**
* **Feature importance (from model gradients or attention)**

## 6. Conclusion and Future Extensions

**This project demonstrated that Transformer architectures can effectively model short-term financial returns from high-frequency data, particularly when enhanced with rich feature engineering and custom loss functions that reflect economic objectives.**

**Key findings:**

* **Modeling across time and across stocks are complementary approaches.**
* **Directional accuracy can be improved through volatility-aware filtering.**
* **Transformers can outperform traditional models in short-horizon prediction under noise.**

**Potential extensions:**

* **Add reinforcement learning for trading signal optimization.**
* **Integrate market microstructure features (order book depth, spread).**
* **Extend training window to test robustness across months.**
* **Explore pretraining on longer history with self-supervised objectives.**

**Introduction**

In the context of algorithmic trading and quantitative finance, accurately forecasting short-term returns can unlock alpha opportunities—even in highly efficient markets. This project applies machine learning techniques—specifically an encoder-only Transformer architecture—to high-frequency stock market data, with the goal of predicting short-term returns for volatile equities over the month of December.

We begin by constructing benchmark models using traditional regularization techniques such as Ridge and Lasso regression. We then proceed to our main contribution: a Transformer-based model tailored to cross-sectional and time-series structures inherent in high-frequency financial data. The ultimate objective is to develop a robust prediction model that maximizes profitability by minimizing both **directional error** and **return deviation**, under realistic market constraints.

This leads us to our guiding research question:

**Can an encoder-only Transformer model capture the complex dependencies across time and assets in high-frequency data to generate economically valuable return predictions?**

**Data preprocessing and feature engineering**

The dataset covers the period from **December 1st to December 31st**, consisting of high-frequency observations recorded every **10 minutes** throughout each trading day. The key columns include:

* *DATE*: Calendar date in YYYYMMDD format
* *SYMBOL*: Unique stock identifier
* *TIME*: Intraday timestamp in HH:MM:SS format
* *MID\_OPEN*: Mid-price (average of best bid and best ask) at the given timestamp
* *ALL\_EX*: Exchange activity indicator

**Preprocessing Pipeline**

1. **Timestamp Indexing**:  
   A datetime column is created by merging DATE and TIME, which is then used as the index for time-series operations.
2. **Return Computation**:  
   Returns are computed as **percentage changes** over 10-minute intervals. We deliberately **exclude overnight returns** and transitions between trading days to avoid distortions from market closures and openings.
3. **Stock Filtering**:  
   Stocks are classified based on their **volatility**, and only the top-**N most volatile assets** are retained. We also define **risk categories** based on recent return variance.
4. **Target Variable**:  
   A target\_returns column is created and **normalized** (mean = 0, std = 1) for model stability. The mean and standard deviation are stored to later **denormalize predictions**.
5. **Feature Normalization**:  
   All other continuous features are also normalized.

**Feature Engineering**

To overcome the limitations of having few raw variables, we generate a range of derived indicators:

Financial indicators

* 1. Momentum
  2. Rolling mean and standard deviation
  3. Skewness and kurtosis
  4. Cumulative volatility
  5. Lagged returns

*[Formulas and rolling windows are detailed in the appendix]*

Non-linear transformations:  
High-order polynomial interactions and non-linear transformations are created to better capture complex market dynamics. *[Formulas are detailed in the appendix]*

Temporal categorical features:

1. Day of the week
2. Name of the day (e.g., Monday)
3. Hour and minute (treated as cyclical variables)

**Predictive Model: Transformer Architecture**

Input Format

After splitting the dataset at the cutoff date '2021-12-18', we create train\_dataset and test\_dataset test tensors. These are organized into DataLoaders with the shape: [B, T, S, D]  
Where:

B = batch size

T = number of lags (sequence length)

S = number of stocks

D = number of features

Due to the high dimensionality of this structure, **memory efficiency is a concern**, which motivates restricting the number of stocks used in training.

Embedding, Cyclical Encoding and Projection

The feature embeddings are handled as follows:

* Categorical Features: symbol, day, and day\_name are embedded into 4-dimensional vectors.
* Time Features: hour and minute are embedded **cyclically** (e.g., via sine and cosine) to capture periodic patterns in trading activity.
* Continuous Features: After embeddings, all inputs are projected into a common space of dimension d\_model before applying positional encoding.

**Attention Mechanisms**

We implement two modes of attention:

* **Cross-sectional Attention (across stocks at a fixed time)**:  
  Input reshaped as [B, T, S, D] → [B \* T, S, D].  
  This allows the model to learn interactions between stocks at each time step but is computationally expensive.
* **Time-series Attention (within stock sequences)**:  
  Input reshaped as [B, T, S, D] → [B \* S, T, D].  
  This applies temporal attention per stock and is more memory-efficient.

**Transformer Block**

Each block consists of:

* **Multi-head self-attention**:  
  Captures dependencies across tokens (either time steps or stocks)
* **Layer normalization**:  
  Applied before and after the attention + feedforward sublayers
* **Feedforward layer** with **GELU activation**:  
  GELU is preferred over ReLU in regression tasks because:
  + It produces smoother gradients
  + It softly handles negative inputs
  + It empirically improves performance in deep networks  
    *[cf "Hendrycks & Gimpel (2016): Bridging Nonlinearities and Stochastic Regularizers with GELUs"]*
* **Dropout and residual connections** are applied to stabilize and regularize training.

**Optional Wrapper**

An optional second round of attention can be applied, again either across time or across stocks. While this can enhance representational power, it greatly increases memory demands and is therefore used selectively.

**Prediction Head and Custom Loss Function**

The model outputs a **return prediction** for each stock and each 10-minute interval in the test set. Since our end goal is to **build a trading strategy**, we care about two objectives:

* **Accuracy of predicted return magnitudes** (low deviation)
* **Correct sign of return** (directional accuracy)

Using a standard loss (e.g., MSE or Huber) results in conservative predictions clustered around zero, due to the low absolute values of returns. To overcome this, we designed a **custom loss function**:

Where:

* MSE: penalizes magnitude
* SignLoss: penalized sign mismatches

This combined loss encourages the model to produce **economically meaningful outputs** while remaining statistically sound.