Smarter Spread Trading with Deep Learning: Building the Future of Automated Investment Decisions

Binqian Chai (bc334)
Yunkai Gao (yg262)
Bohao (Lexi) Yang (by95)
Chenyao Yu (cy230)

STA561D: Probabilistic Machine Learning

Professor Eric B. Laber

April 2025

Contents

Pairs trading system makes investing smarter and easier for everyone

Durham, NC - April 2025 - Today, a team of student researchers from Duke University introduced a new automated tool designed to make pairs trading– a popular market strategy– accessible to more investors. Pairs trading uses statistics to find relationships between stocks or funds, allowing investors to buy and sell in ways that profit from short-term market shifts[2]. Traditionally, it has been a strategy reserved for experts in large institutions. But now, thanks to this new tool, anyone with or without technical backgrounds can get in on the action.

The system analyzes minute-by-minute stock data from 2016 to 2025 in United States and automatically identifies pairs of stocks that are statistically correlated. It can then predict their short-term price movements, which help users to make smarter, faster decisions without needing a deep background in finance or statistics.

What makes this system outstanding is its hybrid approach: it blends traditional financial analysis with cutting-edge deep learning techniques, including a transformer-based architecture that can understand long-term patterns across different stocks from different companies and industries. This means it can spot trading opportunities faster and more reliably than traditional methods.

"Our initial results are highly promising," one team member said. "The system is already capable of delivering stable predictive performance, demonstrating that advanced AI models can meaningfully support real-world trading decisions. Moving forward, we plan to continue improving the system through larger-scale training, model fine-tuning, and extended data coverage. With these enhancements, we aim to push accuracy even higher and unlock new levels of adaptability in changing market environments."

FAQ

Q1. What is the goal of the system?
A1 Our system aims to predict the future movement of the spread between a group of stocks, classifying it into one of three trading signals: decrease (-1), stable (0), or increase (1). Specifically, the spread is considered *stable* if its future average remains within a small tolerance band around the historical mean (typically within $\pm 0.2\,\varepsilon$, where ε denotes the estimated standard deviation of past spread fluctuations). By forecasting spread dynamics based on historical market behavior, the system provides actionable insights for statistical arbitrage strategies in pair trading.

Q2. How is the model designed? What makes it unique?
A2 The model is built on a Transformer-based architecture. Each stock's historical price features and the corresponding spread sequence are encoded separately using dedicated Transformer modules. First, multiple stocks are fused through a stock-level Transformer to capture inter-stock relationships. Then, a cross-attention mechanism fuses the aggregated stock representation with the spread representation, allowing the model to dynamically adjust to both individual stock movements and broader market signals. Unlike traditional methods that rely on fixed assumptions, our design incorporates two layers of fusion, which is well-suited for capturing complex, time-varying relationships.

Q3. What type of data is used for training, and how is it collected?
A3 We use historical stock market data consisting of open, high, low, close prices, and trading volume (OHLCV), along with spread sequences computed between stock pairs. The data is collected using the AlphaVantage API, ensuring access to up-to-date and reliable market information. We further preprocess the data using smoothing techniques such as exponential moving averages (EMA) to highlight meaningful trends.

Q4. How is model performance evaluated?
A4 We evaluate the model on a separate held-out test dataset using classification metrics such as accuracy, precision, recall, and F1 score. In addition to standard evaluation, we simulate trading strategies based on the model's predicted signals and assess portfolio performance metrics such as cumulative returns and maximum drawdown.

Q5 How does the system adapt to changing market conditions?
A5 Our model captures both short-term and long-term dependencies through the Transformer's self-attention mechanism. By incorporating recent spread dynamics directly into the prediction process, the model remains sensitive to shifting market behaviors. Furthermore, because the model has no hard-coded assumptions about mean regression or fixed patterns, it can naturally adapt to changes without human intervention.

Q6 How frequently is the model updated or retrained?
A6 In practice, the model would be retrained periodically, such as every few months, depending on observed market drift. If significant changes in spread behavior or volatility are detected, more frequent updates can be scheduled. The system is designed to easily reload new weights after retraining, supporting continuous learning as new data arrives.

Q7 How do you ensure the model remains robust under volatile conditions?
A7 We apply several strategies to enhance robustness: dropout and weight decay are used to prevent overfitting, while gradient clipping ensures training stability even under extreme input variations. Moreover, the model is trained across a wide range of historical periods, including volatile markets, which exposes it to diverse conditions during training and improves generalization.

Q8 Are there any risks of overfitting to historical data? How is this addressed?
A8 Yes, overfitting is a risk, especially when using complex models on financial data. We address this through early stopping based on validation performance, regularization techniques like weight decay, and dropout within the Transformer blocks. In addition, the model is validated across out-of-sample periods to ensure that it generalizes beyond the specific patterns seen during training.

Q9. What are the biggest advantages compared to traditional pair trading methods?
A9 Traditional pair trading strategies typically rely on using today's spread as a rough proxy for future spread behavior, assuming mean-reversion without explicitly modeling future dynamics. In contrast, our model directly predicts the future spread movement by learning from a richer set of historical features, including multi-stock OHLCV sequences and smoothed spread trends. This allows the system to capture deeper temporal patterns and cross-asset interactions, leading to more informed, dynamic, and robust trading signals compared to static rule-based approaches.

Q10 How does the system scale to larger trading universes compared to traditional methods?
A10 The model's architecture is designed to scale flexibly, as it encodes each stock independently before aggregating information across multiple stocks. However, expanding to much larger stock groups increases both computational complexity and memory requirements. Training with high-frequency data, such as minute-level observations, becomes particularly demanding and would benefit from access to more powerful computational resources, such as high-end GPUs or distributed systems. While the core framework remains applicable at larger scales, future work would focus on optimizing model parallelism and system efficiency to fully leverage available infrastructure and accommodate larger datasets.

Q11. How does the system decide when to short, hold, or go long?
A11 The system monitors the movement of the spread relative to statistically defined entry and exit thresholds based on historical volatility. When the spread exceeds the upper entry limit, it indicates a short position in anticipation of mean reversion. When the spread falls below the lower entry limit, it indicates a long position. When the spread reverts to the mean and crosses the specified exit line, the position is closed. If the spread remains between the entry thresholds, the system recommends holding and no new positions are opened. This rule-based interpretation ensures that trading decisions are based on significant deviations rather than random fluctuations.

Q12 How is the spread between stocks calculated?
A12 In our system, the spread measures how much the price of one stock (called the benchmark stock) differs from a weighted combination of other related stocks (called the hedge stocks). More specifically, at each minute in time, we calculate:

$$s(t) = P_{\text{Benchmark}}(t) - \sum_{i=1}^{n} \beta_i P_{\text{Hedge},i}(t)$$

where $P_{\text{Benchmark}}(t)$ is the price of the main stock we are focusing on at time t, $P_{\text{Hedge},i}(t)$ is the price of the i-th hedge stock at the same time, and β_i is a hedge ratio, a number that tells us how much of each hedge stock we should consider in the comparison. These ratios are usually chosen based on historical relationships (for example, from a linear regression) so that the hedge stocks "match" the benchmark stock as closely as possible. The idea is that if two stocks (or groups of stocks) are usually closely related, their spread should stay near a certain range. When the spread becomes unusually large or small compared to its usual behavior, it could signal a trading opportunity because it is likely that the prices will eventually move back together. Intuitively, the spread acts like a thermometer: it tells whether two stocks are maintaining their usual relationship or deviating abnormally– and identifying these deviations forms the basis for making profitable trading decisions in pairs trading strategies.

Q13 What were the hardest parts of building the system?
A13 One of the biggest challenges we encountered was synchronizing thousands of minute-level data points across many different stocks without introducing errors, which was important to ensure that the model was learning from aligned information. Additionally, we had to contend with the high level of noise inherent in financial markets, where random fluctuations can easily obscure real trading signals. Another major difficulty was preventing overfitting, since patterns that appeared strong in one year often failed to generalize to different time periods. Finally, managing memory and computation was a persistent issue, especially when training large Transformer models on high-frequency data, where each additional stock and each extra time step significantly increased resource demands.

Q14 Why is smoothing important in your data?
A14 In financial markets, prices naturally fluctuate minute by minute, often due to random movements that don't reflect meaningful trends. This randomness is what we call noise. If the model were trained directly on raw, unsmoothed data, it could easily become distracted by these small, unimportant changes and struggle to learn broader, useful patterns. To address this, we apply a technique Exponential Moving Average (EMA), which smooths the spread over time. EMA works by giving more weight to recent prices while still considering past values, helping the model focus on significant trends rather than reacting to every small change. While smoothing cannot eliminate all noise, it improves the model's ability to recognize true signals in the data and makes training more stable and effective overall.

Q15 Could this system work for other trading strategies too?
A15 Yes, it could. Even though we built the system for pairs trading, the way it works is very flexible. The model learns patterns from groups of stocks and uses them to make predictions. This same idea could be used for other strategies, like trading a whole group of stocks together (basket trading), finding price mismatches between different industries (sector arbitrage), or even trying to predict big movements in the entire stock market (index forecasting). Because the model is built to handle different stocks and changing patterns, it can be adapted fairly easily to new types of trading tasks.

Q16 What if there's a huge market crash or unexpected event?
A16 The model may initially struggle during a market crash, like any system trained on mostly "normal" data. However, because it doesn't rely on hardcoded rules, it can adapt faster through retraining. In practice, we would rapidly update the model using post-crash data, allowing it to recover and stay relevant even in dramatically changed environments.

Q17 Can users understand why the model made a prediction?
A17 Our system is built using Transformer models, which naturally use internal attention mechanisms to decide which parts of the stock and spread histories are most important when making predictions. While our current version does not output or visualize these attention maps, the model's structure makes it possible to add this feature in the future. By extracting and visualizing attention maps, we could help users better understand what the model focuses on when making decisions, improving the system's transparency and trustworthiness over time.

Q18 Does the model account for trading fees and real-world costs?
A18 Our current model does not directly factor in trading costs like fees or slippage when making its predictions. However, when we evaluate the system's performance during backtesting, we acknowledge that real-world trading involves costs, and we discuss their potential impact. In future work, we plan to include trading costs more formally, either by adjusting the model's decision-making

process or by building cost-awareness directly into the training and evaluation pipeline.

Q19 Can the system work live in the market?
A19 Our current system has been developed and tested using historical stock data, and it is designed in a way that could support real-time trading in the future. The model, once trained, is able to process new data and make predictions quickly enough for live use, especially when accelerated with GPUs. However, at this stage, we have not yet integrated a live market data feed or built an automatic trading execution system. Deploying the system for real-time trading would require additional engineering steps, such as connecting to a brokerage API and handling data stream reliability, but the underlying architecture is well suited for these future extensions.

Q20 What are the future directions for this project?
A20 One important direction is to expand beyond making only short-term predictions by developing models that can forecast how spreads might evolve over different time horizons. Adding risk management features is another key priority, such as automatically adjusting position sizes based on market volatility or setting thresholds for taking profits and cutting losses. We also hope to enrich the system's inputs by incorporating alternative sources of information beyond price and volume data, including financial news sentiment and options trading activity, to help the model better capture broader market dynamics. Finally, moving toward real-world deployment is a major goal. We plan to start by testing the system in paper trading or sandbox environments, where we can monitor its live performance without risking actual capital. These improvements would help make the system more adaptive, more resilient, and more ready for practical use in live markets.

Technical Appendix

1 Data Processing

In statistical arbitrage pair trading, the quality of data preparation plays a critical role in the robustness of trading signals[5]. Therefore, a comprehensive data processing pipeline has been constructed for minute-level equity data to facilitate pair selection and signal generation based on spread dynamics.

1.1 Data Acquisition and Preparation

In our project, we retrieved minute-level data for 50 U.S. equities across 15 industries from 2016 to 2025 via Alpha Vantage API. In order to maintain the uniformity and completeness of our dataset, we first perform per-symbol downloads segmented by year and month. Due to asynchronous trading across tickers, aligning the timestamps of different stocks is a must. Thus we intersect sets of timestamps across all assets $T = \{t_1, t_2, \ldots, t_m\}$ and retain only timestamps for which data from all tickers are present:

$$T_{\text{aligned}} = \bigcap_{i=1}^{n} T_i \text{ ,where } T_i \text{ is the timestamp for stock i}$$

1.2 Computation Process

Now let us clarify the formulas we used and the computation process, which are the basis of data preparation.

The Spread The spread is the relative deviation in price between a base stock and hedging stocks. Assume that all input stocks have a common index (intersection of timestamps), all stock data frames have equal length and matching indices. For each group of stocks $(S_1, S_2, \ldots, S_n)$, the spread $s(t)$ is:

$$s(t) = P_{S_1}(t) - \sum_{i=2}^{n} \beta_i \cdot P_{S_i}(t)$$

where $P_{S_i}(t)$ is the close price of stock S_i at time t, β_i is the hedge ratio. We treat the first stock as the benchmark and the rest are hedged proportionally.

Exponential Moving Average To suppress noise, we apply EMA (Exponential Moving Average) Smoothing method to the spread:

$$\text{EMA}_t = \alpha \cdot s(t) + (1 - \alpha) \cdot \text{EMA}_{t-1} \quad \text{with} \quad \alpha = \frac{2}{N+1}$$

where $s(t)$ is the spread at time t, N is the time span and $\alpha \in (0, 1)$ is the smoothing parameter. Then the smoothed spread EMA_t can be used in further sequence construction and label classification.

Label Classification via Volatility Thresholds Volatility thresholds are used for our classification to label future spread trends. Suppose we have a spread window $s_{t-L+1:t}$, then we compute

$$\mu_t = \frac{1}{L}\sum_{i=0}^{L-1} s_{t-i}, \quad \sigma_t = \sqrt{\frac{1}{L}\sum_{i=0}^{L-1}(s_{t-i}-\mu_t)^2}$$

where L is the length of historical window. Let $\bar{s}_{\text{future}}$ be the mean spread over a future prediction window. Then we created the classification rule:

- $\bar{s}_{\text{future}} > \mu_t + 0.2\sigma_t \Rightarrow$ Upward (label = 2)
- $\bar{s}_{\text{future}} < \mu_t - 0.2\sigma_t \Rightarrow$ Downward (label = 0)
- Otherwise: Sideways (label = 1)

Feature Representation Each time, we have five features [open, high, low, close, volume]. Thus we know the input sequences $X \in \mathbf{R}^{M\times L\times 5}$, where M is number of stocks in the group. The model operates on a windowed time-series tensor paired with its associated label.

1.3 Dataset Implementation

The above steps are integrated as a pipeline, supporting pair specific processing and variable length groups. For each pair, the spread is computed and smoothed with EMA, the sequences of input tensors are generated for future model and labels are created based on trends. Finally, our dataset will be:

- X: shape = $(M, L, 5)$
- spread: shape = $(L,)$
- label: scalar class $\in \{0, 1, 2\}$

Overall, this data processing pipeline provides the structural foundation for our pair trading model and downstream modeling tasks such as backtesting and real-time deployment.

2 Model

2.1 Model Construction

2.1.1 Motivation and Design Considerations

The design of our model is guided by the nature of the pair trading task. Each sample consists of a group of stocks, represented as multivariate time series, along with an associated spread sequence that captures their relative deviation. The model must process not only the dynamics within each stock but also the interaction across stocks and their relationship with the spread. Furthermore, the number of stocks in a group may vary across samples, adding additional complexity.

Traditional statistical arbitrage methods rely heavily on price ratio or cointegration-based pair selection, assuming stable long-term relationships and stationary mean-reversion behavior [3]. These methods are typically static and unable to model the sequential nature of price dynamics or the influence of global signals like spreads. In particular, they assume a fixed equilibrium relationship and do not update predictions dynamically based on recent market activity.

Machine learning approaches such as recurrent neural networks (RNNs) and long short-term memory (LSTM) models have been used to capture temporal dependencies in financial time series [4]. However, they process data sequentially via hidden states $h_t = f(h_{t-1}, x_t)$, making them difficult to parallelize and less efficient for long sequences or large-scale multi-stock modeling [1]. Convolutional neural networks (CNNs) offer efficiency through local receptive fields, but they struggle with modeling long-range dependencies unless extremely deep architectures are used [1].

To address these limitations, our model leverages the Transformer architecture, known for its self-attention mechanism that captures both short-term and long-term dependencies efficiently [8]. Unlike RNNs, which propagate information step-by-step, Transformers compute dependencies between all time steps simultaneously. Given an input sequence $X \in \mathbb{R}^{T \times d}$, the self-attention layer computes:

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

where $Q = XW^Q$, $K = XW^K$, and $V = XW^V$ are the learned projections. This allows the model to learn relationships between all pairs of time steps regardless of distance. As a result, both local fluctuations and global structure in the input sequence can be effectively modeled.

Furthermore, to enhance the model's ability to capture diverse types of temporal patterns, our architecture adopts **multi-head attention** [8]. Instead of relying on a single attention map, multi-head attention projects the input into multiple subspaces, allowing the model to simultaneously capture dependencies at different temporal scales and behavioral patterns. Formally, multi-head attention is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O$$
$$\text{where} \quad \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

This mechanism significantly improves the model's expressiveness, especially in complex financial environments where multiple latent factors may drive asset dynamics.

The theoretical expressiveness of Transformers also supports their adoption in our context. Recent work shows that Transformers are universal approximators of sequence-to-sequence functions under mild assumptions [9], which provides a strong foundation for modeling the nonlinear and dynamic interactions among multiple stock prices and spread signals.
In our application, the model receives input tensors of shape $X \in \mathbb{R}^{B\times M\times L\times 5}$ and $s \in \mathbb{R}^{B\times L}$, where B is the batch size, M is the number of stocks per group, L is the sequence length, and 5 denotes OHLCV features.
The model aims to learn a function that maps input sequences to trading signal classes:

$$f : \mathbb{R}^{M\times L\times 5} \times \mathbb{R}^{L} \to \{0, 1, 2\}$$

that predicts the future movement of the spread based on historical price behavior and its smoothed dynamics. The predictive task is formulated as a three-class classification problem using cross-entropy loss:

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=0}^{2} \nVdash(y_i = c) \log \hat{p}_{i,c}$$

where y_i is the true label, and $\hat{p}_{i,c}$ is the predicted probability for class c.

Overall, by replacing sequential RNN updates with self-attention and incorporating both stock-level and spread-level signals, the Transformer-based architecture offers a more expressive and scalable solution for modeling the dynamics of statistical arbitrage in pair trading.

2.1.2 Component Overview

Our proposed model is composed of five core modules: **stock encoders**, a **spread encoder**, a **stock fusion module**, a **final fusion module**, and a **classification head**. Each module is designed to handle a specific aspect of the pair trading task and contributes to building a flexible and scalable prediction pipeline.

- **Stock Encoders:** Each stock's OHLCV input sequence of shape $L \times 5$ is first passed through a shared linear projection to embed features into a latent space of dimension d_{model}. Then, we add sinusoidal positional encoding [8] to inject time step information into the embeddings, before feeding them into a Transformer encoder:

$$X^{(i)} \in \mathbb{R}^{L\times 5} \quad \to \quad H^{(i)} \in \mathbb{R}^{L\times d_{\text{model}}}$$

This design enables learning temporal dynamics specific to each stock while preserving a shared latent space.

- **Spread Encoder:** The EMA-smoothed spread sequence $s \in \mathbb{R}^{L\times 1}$ undergoes a similar process: linear projection followed by sinusoidal positional encoding, and then a dedicated Transformer encoder to capture the global deviation behavior:

$$s \in \mathbb{R}^{L\times 1} \quad \rightarrow \quad h^{(\text{spread})} \in \mathbb{R}^{L\times d_{\text{model}}}$$

- **Stock Fusion Module:** After encoding each stock individually, all encoded representations are stacked and aggregated along the stock axis. The stacked tensor is passed through a second Transformer encoder to learn inter-stock relationships. The output is then averaged across stocks to obtain a unified representation:

$$F = \text{Transformer}\left(\text{Stack}(H^{(1)}, \ldots, H^{(n)})\right), \quad \bar{F} = \text{Mean}_{\text{stock}}(F)$$

- **Final Fusion Module:** The spread representation attends to the fused stock features via cross-attention, using the spread as the query and the stocks as key/value. This is followed by a self-attention layer and global average pooling:

$$\text{CrossAttn}(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d}}\right)V$$

  where $Q = \text{Spread}, \ K, V = \text{Stocks}$.

- **Classification Head:** The final pooled vector is passed through a two-layer MLP to produce logits for three classes representing the predicted trading signal:

$$\hat{y} = \text{MLP}(z), \quad z = \text{GlobalPool}(\text{SelfAttn}(F))$$

In summary, our model hierarchically encodes time-series information at the stock level, fuses cross-stock dependencies, and incorporates spread signals via attention mechanisms. As illustrated in Figure 1, the architecture is modular and allows generalization to arbitrary combinations of stock pairs and prediction horizons.

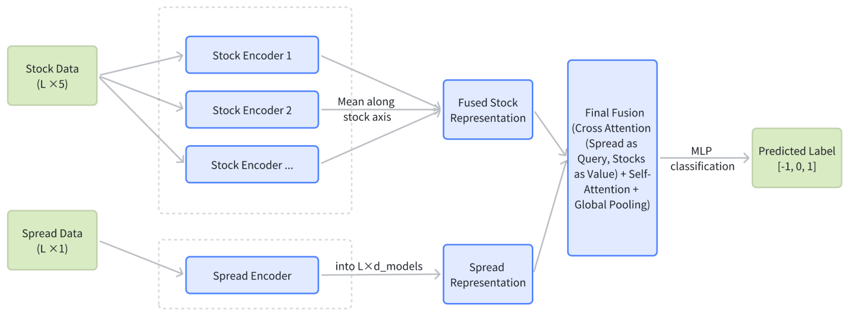

Figure 1: Architecture of the StockFusionTransformer. Each stock is encoded individually, aggregated temporally, and fused with spread features via cross-attention. The final signal is predicted through an MLP into {-1, 0, 1}.

2.1.3 Input and Output Format

The model input is a 4D tensor of shape $(B, M, L, 5)$, where B is batch size, M is number of stocks per group, L is sequence length, and 5 corresponds to OHLCV features. The spread sequence has shape (B, L). The model outputs a probability vector of shape $(B, 3)$, indicating trading actions: *Decrease (-1)*, *Stable (0)*, or *Increase (1)*.

2.2 Model Training

2.2.1 Loss Function

To train the model to classify future spread trends into three categories, decrease (-1), stable (0), and increase (1), we adopt the standard categorical cross-entropy loss. Given the output logits $z \in \mathbb{R}^3$ from the classification head and the one-hot encoded ground truth label $y \in \{0, 1\}^3$, we first apply the softmax function to convert logits into probabilities:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{3} \exp(z_j)} \quad \text{where } z_i \text{ is the unnormalized logit before softmax}$$

The cross-entropy loss is then computed as:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{3} w_i \cdot y_i \log(p_i)$$

Here, w_i denotes the weight for class i, which is used to mitigate class imbalance in the training data. In our baseline implementation, we assign equal weights $(w_i = 1)$ to all classes. However, the training pipeline allows for assigning custom weights based on class frequency if needed.

2.2.2 Optimization and Learning Schedule

We optimize the model using the AdamW optimizer [6], a widely adopted variant of Adam that decouples weight decay from the gradient update. AdamW adapts learning rates for each parameter using first- and second-order moment estimates, while applying L_2 regularization directly to the parameters. Given model parameters θ, the AdamW update rule is:

$$\theta_t = \theta_{t-1} - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_{t-1} \right)$$

where $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected estimates of the first and second moments, and λ is the weight decay coefficient. We use the default values $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

The initial learning rate is set to 5×10^{-5}. To stabilize training and adapt to validation performance, we employ a 'ReduceLROnPlateau' scheduler. The scheduler monitors the validation loss and reduces the learning rate by a factor of 0.5 if no improvement is observed for 50 consecutive epochs. The learning rate is bounded below by 10^{-6}.

This dynamic learning rate adjustment helps accelerate convergence during early training and enables fine-tuning as the model approaches a local minimum, while AdamW regularization improves generalization by preventing overfitting.

2.2.3 Regularization and Early Stopping

To improve model generalization and prevent overfitting, we apply several regularization techniques during training. First, dropout with a rate of 0.1 is used in both the Transformer encoders and the classification head. This encourages the model to learn redundant representations by randomly dropping units during training, improving robustness [7]. Second, we apply L_2 weight decay with a coefficient of 10^{-4} in the Adam optimizer to penalize large parameter values and promote model simplicity. Additionally, to mitigate gradient explosion—particularly in attention modules, we then use gradient clipping with a maximum norm of 1.0 during backpropagation.

The early stopping is introduced to avoid overtraining and improve computational efficiency. After each epoch, the validation loss is monitored and training is terminated if no improvement is observed for 100 epochs in a row. This strategy helps prevent overfitting of the training data and ensures that the final model checkpoint corresponds to the best performing parameters on the validation set.

3 Evaluation & Results

In this section, we present a detailed evaluation of our model's predictive performance on unseen test data. We assess not only the overall classification accuracy but also how well the model distinguishes between different trading signals: Decrease, Stable, and Increase. We further analyze the results through a confusion matrix, ROC curves, precision-recall curves, training dynamics, and dataset characteristics, and we discuss key strengths and limitations observed during testing.

3.1 Evaluation Setup

The model was trained for a maximum of 500 epochs. Early stopping, as described in Section 2.2.3, was applied based on validation loss, with a patience of 100 epochs. Training was terminated automatically, and the best model checkpoint was selected at epoch 224, corresponding to the lowest validation loss observed. Evaluation was conducted on a separate held-out test set consisting of stock pairs and time periods that were never seen during training or validation. We report standard classification metrics, including accuracy, precision, recall, and F1-score for each trading signal.

3.2 Classification Performance

The model achieved an overall test accuracy of 72.21% at the best checkpoint. As summarized in Table 1, the system performs well in identifying directional spread movements. It achieves an F1-score of 0.75 for predicting "Decrease" signals and 0.72 for "Increase" signals. However, performance on "Stable" signals is notably poor, likely due to both class imbalance and the inherent difficulty of detecting periods without significant price divergence.

Class	Precision	Recall	F1-score	Support
Decrease (-1)	0.70	0.81	0.75	71,693
Stable (0)	0.00	0.00	0.00	6,295
Increase (1)	0.75	0.69	0.72	69,724
Overall Accuracy		72.21%		

Table 1: Classification report on the test set.

The confusion matrix shown in Figure 2 provides additional insights into the model's behavior. The model shows strong performance distinguishing "Decrease" and "Increase" signals, while predictions for "Stable" signals are rare and often misclassified as directional changes, indicating that truly stationary spread behavior is harder for the model to recognize. This limitation reflects the reality of financial markets, where truly stable periods are uncommon, and spread dynamics are often dominated by movement.

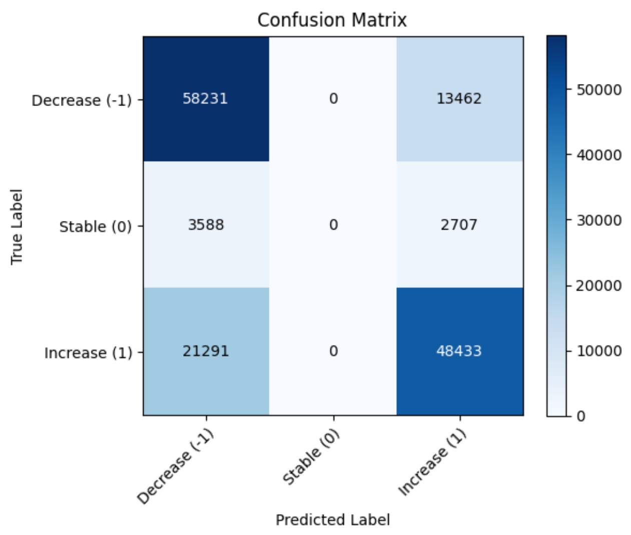

Figure 2: Confusion matrix on the test set. The model performs well on directional signals but struggles with stable signals.

3.3 Class Imbalance Analysis

The class distribution of the test dataset is visualized in Figure 3. The figure reveals a severe imbalance across classes: while the "Decrease" (-1) and "Increase" (1) categories each have over 70,000 samples, the "Stable" (0) category has fewer than 7,000 samples. This disparity explains why the model struggles to predict stable spread behavior, as there were significantly fewer training examples for this class. Addressing this imbalance could potentially improve the model's ability to recognize periods of market stability.

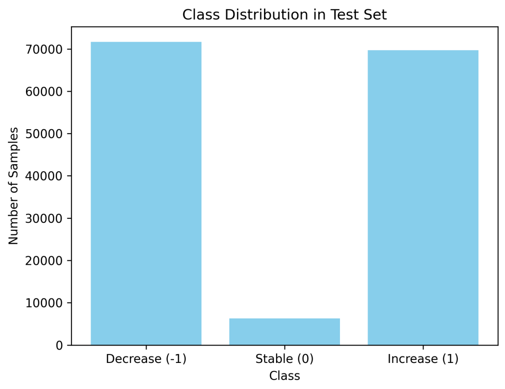

Figure 3: Class distribution in the test dataset. The "Stable" class (0) is significantly underrepresented compared to "Decrease" (-1) and "Increase" (1) classes. This severe class imbalance likely contributes to the model's difficulty in accurately predicting stable spread behavior.

3.4 Model Discrimination Ability

Although ROC curves are most naturally suited for binary classification, we generated one-vs-rest ROC curves for each class to evaluate discriminative ability. As shown in Figure 4, the model achieves strong separation for "Decrease" and "Increase" classes, with AUC scores around 0.74. The "Stable" class, by contrast, yields an AUC close to 0.50, reflecting limited predictive ability for stationary behavior. Predicting "Stable" signals remains the most challenging, consistent with other evaluation metrics.

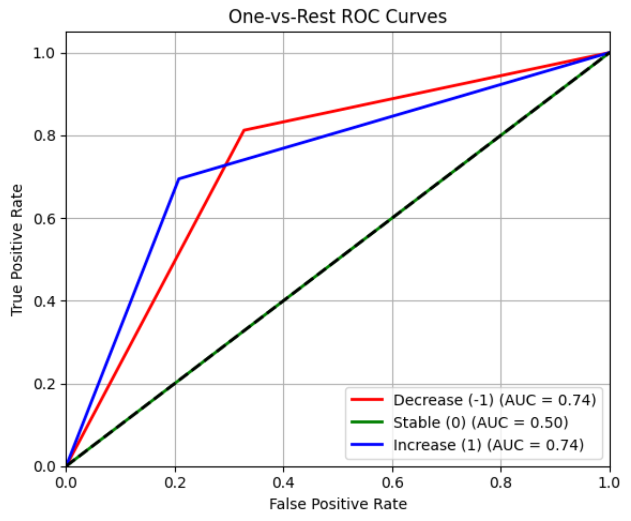

Figure 4: One-vs-rest ROC curves for each trading signal class.

We further plot the Precision-Recall (PR) curves in Figure 5, which provide a more informative evaluation for imbalanced classification tasks compared to ROC curves. The PR curves reveal that the model achieves relatively high precision and recall when predicting directional spread movements (Decrease and Increase classes), demonstrating its ability to identify actionable trading opportunities. In contrast, the PR curve for the Stable class is significantly lower and declines almost linearly, indicating that the model struggles to confidently detect periods of market stability. This performance gap aligns with the severe class imbalance observed in the test set and highlights the need for targeted strategies to improve Stable class detection in future work.

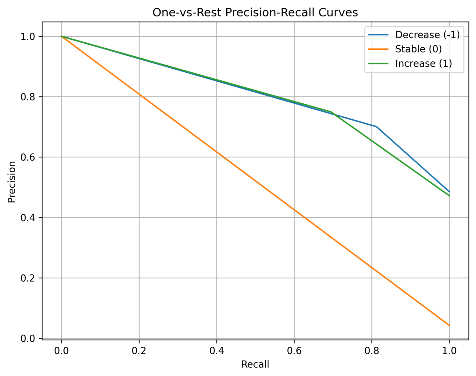

Figure 5: One-vs-rest Precision-Recall (PR) curves for each trading signal class. The model maintains relatively high precision and recall for “Decrease” (-1) and “Increase” (1) classes, while performance on the “Stable” (0) class is significantly lower, reflecting the difficulty in accurately detecting periods of market stability under severe class imbalance.

3.5 Training Dynamics

The training and validation loss curves are visualized in Figure 6. As shown, both losses decrease steadily during early epochs, indicating effective learning. After approximately 300 epochs, the validation loss flattens around a value of 0.61–0.62, and no significant further improvements are observed. According to our early stopping criterion described in Section 2.2.3, training was terminated automatically after 100 consecutive epochs without improvement, selecting the best checkpoint at epoch 224. This stopping behavior prevents unnecessary overtraining, reduces computational time, and ensures that the final model generalizes well to unseen data without overfitting.

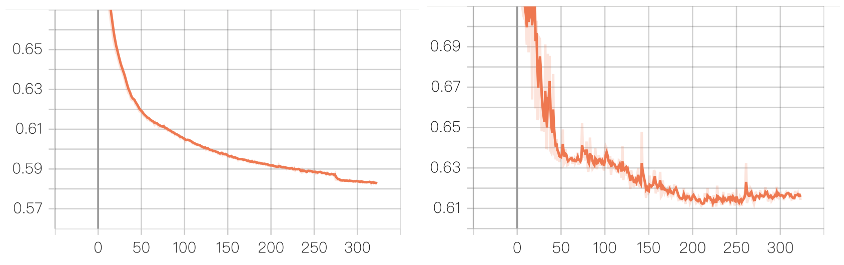

Figure 6: Training loss (left) and validation loss (right) over epochs. Validation loss stabilizes after approximately 300 epochs.

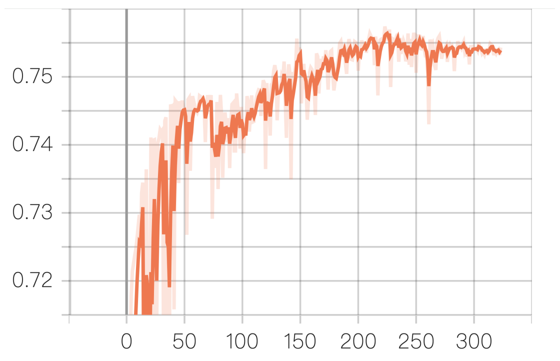

Figure 7: Validation accuracy over epochs. The trend confirms good generalization without overfitting.

3.6 Discussion

Overall, the results indicate that our system effectively captures the most actionable trading signals– directional spread movements, while encountering expected challenges in detecting stable market conditions. For pairs trading strategies that primarily rely on identifying opportunities to open long or short positions, the model's predictions can provide meaningful support to decision-making. However, the difficulty in accurately predicting "hold" signals underscores the limitations imposed by severe class imbalance and the subtlety of neutral spread dynamics. As a result, users seeking to optimize trading strategies that heavily depend on stable market periods may require further model refinements.

3.7 Future Improvements

Future work will focus on addressing the model's current limitations, particularly its difficulty in stable spread prediction. Potential directions include rebalancing the dataset to mitigate class imbalance, refining label definitions to better differentiate marginal spread movements, and incorporating specialized techniques for rare event detection. Additionally, integrating transaction cost modeling into the evaluation process would enable a more realistic assessment of net trading profitability. Expanding the system to leverage alternative data sources, such as sector indices, macroeconomic indicators, or sentiment analysis, could further enhance robustness and adaptability in dynamic market environments.

4 Conclusion

In summary, our system demonstrates the feasibility and promise of leveraging Transformer-based architectures for intelligent pairs trading. By effectively identifying directional spread opportunities while maintaining strong generalization, the model provides actionable insights for investors seeking to automate trading strategies. However, several limitations remain: the model struggles to accurately detect stable market conditions due to severe class imbalance, and real-world factors such as transaction costs and slippage are not yet incorporated into the decision process. Future work will focus on addressing these challenges by rebalancing training data, refining label definitions, and extending the framework to directly optimize risk-adjusted returns. With targeted improvements and the integration of broader financial signals, the system has the potential to evolve into a robust decision-support tool for both individual traders and institutional investment platforms.

Code Availability: The source code, data processing scripts, and model implementation for this project are available at: `https://github.com/buishuohua/PT_Unsupervised`

References

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

[2] Robert J Elliott, John Van Der Hoek*, and William P Malcolm. “Pairs trading”. In: *Quantitative Finance* 5.3 (2005), pp. 271–276.

[3] Evan Gatev, William N Goetzmann, and K Geert Rouwenhorst. “Pairs trading: Performance of a relative-value arbitrage rule”. In: *The Review of Financial Studies* 19.3 (2006), pp. 797–827.

[4] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[5] Christopher Krauss. “STATISTICAL ARBITRAGE PAIRS TRADING STRATEGIES: REVIEW AND OUTLOOK”. In: *Journal of Economic Surveys* 31.2 (2017), pp. 369–659. DOI: `https://doi.org/10.1111/joes.12153`.

[6] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[7] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research*. 2014.

[8] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[9] Chulhee Yun et al. “Are transformers universal approximators of sequence-to-sequence functions?” In: *International Conference on Learning Representations*. 2020.