Investments: Machine Learning for Finance

Xueying Cheng, Dimitry Anishchenko, Kenta G. Frei◇

***Abstract*—This project investigates the effectiveness of deep learning architectures—CNN, RNN, DNN, and autoencoders (AE)—in forecasting financial market returns using macro-financial indicators. We systematically select feature combinations, preprocess time series data, and optimize network configurations to identify the most predictive signal sets. Our experiments reveal that RNNs with engineered features like [CPI, _TY, _LCP] achieve strong Sharpe ratios, while CNNs demonstrate stable learning and temporal pattern extraction. Overall, the results confirm that model-specific architectures combined with thoughtful signal design yield robust predictive performance in financial time series modeling.**

I. Introduction

Machine learning methods, especially those based on neural networks, offer a promising framework for learning predictive structures directly from raw financial signals. By relaxing the assumptions of linearity and allowing for complex feature transformations, these methods can potentially extract more information from available data. In particular, deep learning architectures have shown impressive results in areas such as image recognition and natural language processing, and their application to financial time series has become an active area of research.

This paper contributes to the growing literature at the intersection of machine learning and finance by empirically exploring the use of deep learning techniques in building and evaluating trading signals. Rather than relying on fixed economic assumptions, we adopt a data-driven approach aimed at understanding how different neural network models perform when applied to real financial data in the context of active portfolio management.

II. Objective

The primary objective of this project is to develop a machine learning–based trading algorithm for a stock index, using historical financial signals as inputs. The algorithm aims to learn patterns in the data that are predictive of future returns and to translate these predictions into actionable investment strategies.

By training neural network models to predict future asset performance—whether at the level of individual stocks, factor portfolios, or the index itself—we aim to construct trading strategies that systematically exploit learned patterns. The performance of these strategies will be evaluated based on both predictive power (measured through in- and out-of-sample information coefficients) and risk-adjusted returns (measured through the Sharpe ratio).

The approach is entirely data-driven and does not assume any specific economic structure a priori. This allows us to test whether machine learning models can identify exploitable signals purely from historical observations, and whether these signals can consistently translate into profitable trades across different network architectures and signal sets.

◇ Master of Science UZH ETH in Quantitative Finance

III. Network Architectures

A. Deep feedforward neural network

We implement a deep feedforward neural network (DNN) to capture nonlinear relationships between financial signals and asset returns. The model structure and regularization choices are motivated by the need to balance expressiveness and overfitting control in a limited-sample financial setting.

Let $\mathbf{x}_t \in \mathbb{R}^d$ denote the input feature vector at time t, where $d \in \{2, 3, 4\}$ corresponds to the number of selected signals. The DNN architecture consists of the following layers:

- **Input layer:** Receives standardized signals using z-score normalization.
- **Hidden layer 1:**

$$\mathbf{h}^{(1)} = \text{ReLU}(\mathbf{W}^{(1)}\mathbf{x}_t + \mathbf{b}^{(1)}),$$

  followed by batch normalization and a dropout layer with rate $p = 0.2$, where $\mathbf{W}^{(1)} \in \mathbb{R}^{64\times d}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{64}$.
- **Hidden layer 2:**

$$\mathbf{h}^{(2)} = \text{ReLU}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}),$$

  followed by a dropout layer with rate $p = 0.2$, where $\mathbf{W}^{(2)} \in \mathbb{R}^{32\times 64}$, $\mathbf{b}^{(2)} \in \mathbb{R}^{32}$.
- **Output layer:**

$$\hat{y}_t = \mathbf{w}^{(3)} \cdot \mathbf{h}^{(2)} + b^{(3)},$$

  where $\hat{y}_t \in \mathbb{R}$ is the predicted return.

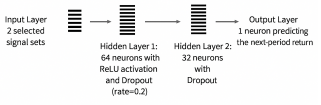


Fig. 1: DNN Architecture

To mitigate overfitting in our deep neural network model, particularly given the limited sample size of macro-financial data, we apply the following regularization strategies:

1) **Dropout:** We apply a dropout layer with a rate of 0.2 after each hidden layer to prevent overfitting. The first dropout follows a batch normalization layer after the first hidden layer, while the second dropout directly follows the second hidden layer.
2) **Early Stopping:** The model monitors validation loss and applies early stopping if no improvement is observed over 5 consecutive epochs. The best-performing weights are restored to avoid overfitting.
3) **Standardization:** All input signals are standardized using z-score normalization (zero mean and unit variance) based on the training set. This preprocessing ensures numerical stability and equal feature contribution.

B. Autoencoders

Autoencoder learns an approximate identity function, but through a bottleneck that forces the network to focus on the most relevant features of the data.

An autoencoder consists of two main components:

- **Encoder:** maps a high-dimensional input to a compressed latent representation (embedding). Formally, for an input $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{z} = f_\theta(\mathbf{x}) \in \mathbb{R}^m \quad \text{with } m < n$$

- **Decoder:** reconstructs the input $\mathbf{x}$ from the latent code $\mathbf{z}$:

$$\hat{\mathbf{x}} = g_\phi(\mathbf{z}) = g_\phi(f_\theta(\mathbf{x}))$$

The functions f_θ and g_ϕ are implemented as neural networks (typically feedforward), with parameters θ and ϕ learned during training.

The training objective of the autoencoder is to minimize the discrepancy between the original input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$. A common metric is the *mean squared error (MSE)*:

$$\mathcal{L}_{\text{AE}}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{x} - g_\phi(f_\theta(\mathbf{x}))\|^2$$

This loss is minimized using gradient descent or one of its variants (e.g., Adam, RMSProp).

The *Contractive Autoencoder* (CAE) is an extension of the standard autoencoder designed to learn **robust and invariant representations** by explicitly penalizing the sensitivity of the encoder to small variations in the input data. While a standard autoencoder minimizes reconstruction error, it may still learn latent codes that vary significantly even for minor perturbations of the input. The CAE addresses this by introducing a *regularization term* that contracts the encoding function around the training examples.

This is achieved by penalizing the *Jacobian matrix* of the encoder function with respect to the input, which measures how much the latent representation $\mathbf{z} = f_\theta(\mathbf{x})$ changes in response to small perturbations of $\mathbf{x}$.

The CAE modifies the standard reconstruction loss by adding a contractive penalty:

$$\mathcal{L}_{\text{CAE}} = \mathcal{L}_{\text{reconstruction}} + \lambda \left\| J_{f_\theta}(\mathbf{x}) \right\|_F^2$$

Where:

- $\mathcal{L}_{\text{reconstruction}} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$ is the mean squared reconstruction error.
- $J_{f_\theta}(\mathbf{x})$ is the Jacobian matrix of partial derivatives of the encoder output $\mathbf{z}$ with respect to the input $\mathbf{x}$.
- $\| \cdot \|_F^2$ denotes the *squared Frobenius norm*.
- λ is a hyperparameter controlling the strength of the regularization.

For a simple encoder layer with activation function $h = \sigma(Wx + b)$, the Jacobian can be written as:

$$J_{f_\theta}(\mathbf{x}) = \text{diag}(\sigma'(Wx + b)) \cdot W$$

Where σ' is the derivative of the activation function, applied element-wise.

The *Variational Autoencoder* (VAE) is a generative extension of the autoencoder that incorporates concepts from **probabilistic graphical models** and **variational inference**. Unlike standard autoencoders, which learn a deterministic mapping from inputs to latent codes, the VAE learns a **distribution over the latent space**, enabling it to generate new data points by sampling from this learned distribution.

In a VAE, we assume that the observed data $\mathbf{x}$ is generated from a set of latent variables $\mathbf{z}$ through the following process:

1) Sample $\mathbf{z} \sim p(\mathbf{z})$, where $p(\mathbf{z})$ is a prior distribution, usually $\mathcal{N}(0, I)$.
2) Generate $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$, where p_θ is the decoder network.

The encoder approximates the true posterior $p(\mathbf{z}|\mathbf{x})$ with a tractable distribution $q_\phi(\mathbf{z}|\mathbf{x})$, typically modeled as a multivariate Gaussian:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}(\mathbf{x}), \text{diag}(\boldsymbol{\sigma}^2(\mathbf{x})))$$

where $\boldsymbol{\mu}(\mathbf{x})$ and $\boldsymbol{\sigma}^2(\mathbf{x})$ are the outputs of the encoder network.

1) Loss Function (ELBO): The VAE is trained by maximizing the *Evidence Lower Bound* (ELBO) on the marginal log-likelihood:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

The first term is the expected log-likelihood of the reconstruction (reconstruction loss), and the second term is the *Kullback–Leibler divergence*, which regularizes the latent space by encouraging the approximate posterior to match the prior distribution.

2) Reparameterization Trick: To enable backpropagation through the sampling process, the *reparameterization trick* is used:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$

This reformulation allows gradients to flow through $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, making the optimization tractable via standard stochastic gradient descent.

C. Convolutional Neural Networks (CNNs)

To capture temporal dependencies in macro-financial time series data, we employ a one-dimensional Convolutional Neural Network (CNN). CNNs are known for their ability to extract translation-invariant and local patterns, making them suitable for financial time series forecasting.

Let $\mathbf{X}_t \in \mathbb{R}^{L \times d}$ denote the input matrix at time t, where L is the sequence length (i.e., time window size), and $d = 3$ represents the number of input features. Each row $\mathbf{x}_i \in \mathbb{R}^d$ corresponds to a vector of standardized macro-financial indicators.

The CNN model architecture consists of the following components:

```
Input:
[IR, _TY, _LCP] over 45 time steps
→ Shape: (128, 45, 3)
Layers:
Conv1D(32, kernel=3) → ReLU → Dropout(0.2)
Conv1D(64, kernel=5) → ReLU → Dropout(0.4)
Conv1D(128, kernel=7) → ReLU → Flatten
Output:
Dense(1) → Predicts next-period '_MKT' return
```

- **Input Layer:** Takes the sequence $\mathbf{X}_t \in \mathbb{R}^{L \times d}$ as input. All features are standardized using z-score normalization to ensure stable training.
- **Convolutional Layers:** Three 1D convolutional layers with causal padding are applied to extract temporal features:

$$\mathbf{H}^{(1)} = \text{ReLU}(\text{Conv1D}(\mathbf{X}_t; \mathbf{W}_1, k_1))$$
$$\mathbf{H}^{(2)} = \text{ReLU}(\text{Conv1D}(\mathbf{H}^{(1)}; \mathbf{W}_2, k_2))$$
$$\mathbf{H}^{(3)} = \text{ReLU}(\text{Conv1D}(\mathbf{H}^{(2)}; \mathbf{W}_3, k_3))$$

  where k_i denotes the kernel size and $\mathbf{W}_i$ the filter weights. Each convolutional layer is followed by **Batch Normalization** and **Dropout** (e.g., $p = 0.2$).
- **Regularization:**
  - **Dropout:** Applied after each convolutional layer to prevent overfitting by randomly deactivating neurons during training.
  - **L2 Regularization:** All convolutional layers are penalized with an ℓ_2-norm:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{loss}} + \lambda \sum_i \|\mathbf{W}_i\|_2^2$$

  where $\lambda = 0.003$ controls the strength of the regularization.
  - **Early Stopping:** Training is stopped if validation loss does not improve over 3 consecutive epochs to avoid overfitting. **MaxPooling** is added after each Conv1D to downsample the temporal dimension and reduce overfitting. This simulates temporal abstraction, helping the model focus on broader trends instead of noise.
- **Fully Connected Layers:** After flattening, the time-dependent features are passed to a dense layer with ReLU activation, followed by a final output neuron:

$$\hat{y}_t = \sigma(\mathbf{w}^\top \mathbf{h} + b)$$

  where $\sigma(\cdot)$ is the sigmoid function, yielding a probability score for a positive return.

D. Recurrent Neural Networks (RNNs)

To model the temporal structure in macro-financial signals, we employ a stacked Long Short-Term Memory (LSTM) neural network. The model is designed to process sequential data and capture dynamic dependencies across time.

Let $\mathbf{X}_t \in \mathbb{R}^{L \times d}$ denote the input sequence at time t, where $L = 20$ is the sequence length (i.e., lookback window) and $d = 3$ is the number of selected input features. The architecture is as follows:

- **Input:** A sequence $\mathbf{X}_t \in \mathbb{R}^{20 \times 3}$, consisting of standardized features over the past 20 days.
- **LSTM Layers:** The core of the model consists of three stacked LSTM layers with either decreasing (128-64-32) or constant (64-64-64) hidden units:

$$\mathbf{h}^{(i)} = \text{LSTM}_i(\mathbf{h}^{(i-1)}), \quad i = 1, 2, 3$$

  where $\mathbf{h}^{(0)} = \mathbf{X}_t$. Each LSTM layer captures different levels of temporal abstraction.
- **Dropout:** Dropout is applied after the first and second LSTM layers to regularize the model:
  - Configuration 1: $p_1 = 0.1$, $p_2 = 0.3$
  - Configuration 2: $p_1 = 0.2$, $p_2 = 0.4$
- **Dense Layers:** The final LSTM output is passed to a dense layer (e.g., 128 or 256 neurons, ReLU activation), followed by a sigmoid output unit:

$$\hat{y}_t = \sigma(\mathbf{w}^\top \mathbf{h}^{(3)} + b)$$

  The output $\hat{y}_t \in (0, 1)$ represents the predicted probability of a positive return.
- **Loss and Optimization:** The model is trained using the binary cross-entropy loss and the Adam optimizer with learning rate $\eta = 0.001$. Early stopping is employed to terminate training when validation loss does not improve for 2 consecutive epochs.
- **Regularization:** All LSTM layers are penalized with L2 regularization to mitigate overfitting:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{bce}} + \lambda \sum_i \|\mathbf{W}_i\|_2^2, \quad \lambda = 0.001$$

IV. Experiments

A. DNN

1) Choice of signals: We selected macro-financial indicators based on their predictive power measured by rolling Spearman rank Information Coefficient (IC). Using a 60-period rolling window, we computed the average IC between each candidate signal and next-day market returns. The top 10 features (fig. 2) with the highest average rolling ICs were selected: `['DY', 'YSS', 'MOV', '_TY', 'M2', 'UN', '_AU', 'STP', '_DXY', 'MG']`.

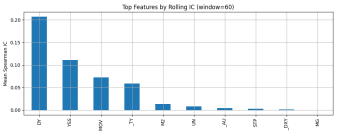


Fig. 2: Top Features

We then generated combinations of 2, 3, and 4 signals and evaluated each set using a DNN trained on the selected features. The top combinations are summarized below:

TABLE I: Top Signal Sets by Spearman IC and Sharpe Ratio

Signal Set	Spearman IC	Sharpe Ratio
(DY, M2, UN)	0.126	**1.94**
(DY, M2, _AU, STP)	0.121	-4.88
(DY, _AU, _DXY, MG)	0.118	-3.62
(DY, YSS, _DXY)	0.111	**4.29**
(DY, UN, _AU, STP)	0.109	3.41
(YSS, M2, _DXY)	0.099	2.33

We tried these sets with DNN model to see their performance, and then we selected the following two signal sets for deeper analysis and strategy modeling:

- (DY, M2, UN): High IC and moderate Sharpe
- (DY, YSS, _DXY): Moderate IC but excellent Sharpe performance

And we have these two hypothesis:

Hypothesis 1: This signal combination includes dividend yield (DY), money supply growth (M2), and unemployment rate (UN), all of which are classical macroeconomic indicators with known predictive power. We hypothesize that this signal set should exhibit consistent predictive ability over time and yield a stable long-short trading strategy with positive Sharpe ratio.

Hypothesis 2: This combination blends dividend yield (DY) with a yield spread signal (YSS) and the inverse dollar index (_DXY). The intuition is that interest rate differentials (YSS) and currency strength (_DXY) indirectly influence capital flows and equity valuations. We expect moderate predictive power with potential regime-dependent performance.

2) Choice of network type and parameters: We employ a deep neural network (DNN) to model the non-linear relationship between selected macro-financial signals and future market returns. The network architecture is deliberately kept simple and shallow, due to the relatively small dataset and the need to prevent overfitting. The architecture consists of the following components:

- **Input Layer:** The input dimension corresponds to the number of selected macro-financial signals (typically 3). All input features are standardized using z-score normalization.
- **Hidden Layers:** The model consists of two fully connected hidden layers. The first hidden layer has 64 neurons followed by Batch Normalization and a Dropout layer with a dropout rate of 0.2. The second hidden layer has 32 neurons with ReLU activation, followed by another Dropout layer (dropout rate 0.2).
- **Output Layer:** A single neuron with linear activation is used to predict the expected return.
- **Training Strategy:** The model is trained with the Adam optimizer for a maximum of 100 epochs and a batch size of 32. Early stopping is applied with a patience of 5 based on validation loss, and the best model weights are restored to reduce overfitting.

3) Results: To evaluate the predictive power and economic value of the selected signal sets, we trained deep feedforward neural networks (DNNs) on two high-ranking combinations from the Spearman IC ranking: Signal Set A: (DY, M2, UN) and Signal Set B: (DY, YSS, _DXY).

The predictive performance was assessed using the Spearman Information Coefficient (IC), Signal Set A achieved an out-of-sample IC of 0.666, indicating strong rank correlation and robust signal quality. ; Signal Set B yielded a lower IC of 0.569, but still above noise level, showing decent predictive structure. The time-series plots of predicted signals versus true returns demonstrate that both models capture directional trends over time, though Signal Set A generally aligns more closely with realized returns.

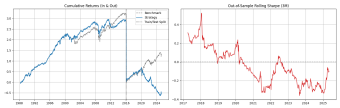


Fig. 3: ('DY', 'M2', 'UN')

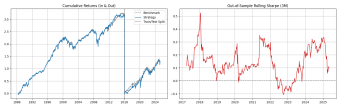


Fig. 4: ('DY', 'YSS', '_DXY')

We evaluate two macroeconomic signal combinations using a deep neural network (DNN)-based long-short equity strategy, trained on data prior to 2016 and tested out-of-sample from 2016 to 2025. The key metrics are IC (Information Coefficient), Sharpe Ratio, and visual inspection of cumulative returns and rolling Sharpe.

The evaluation results show that the first signal set (DY, M2, UN) fails to deliver robust out-of-sample performance, with low information coefficient (IC = 0.064) and near-zero Sharpe

ratio (SR = 0.05), indicating limited predictive power and poor risk-adjusted returns after 2016. In contrast, the second signal set (DY, YSS, _DXY) performs more consistently out-of-sample, achieving a higher Sharpe ratio (SR = 0.11) despite a negligible IC. Its cumulative returns track the benchmark closely, and the rolling Sharpe ratio remains positive in several subperiods, especially post-2020. This suggests the second set may capture regime-dependent dynamics not fully reflected in linear correlation.

Hypothesis 1: Hypothesis 1 is not supported. While the signal set is grounded in economic rationale, it lacks robustness in the out-of-sample period. The model likely overfits to pre-2016 dynamics and fails to generalize.

Hypothesis 2: Hypothesis 2 is partially supported. Although the IC is weak, the strategy captures certain regime-dependent patterns—especially post-2020—which leads to competitive risk-adjusted returns

To address these shortcomings, several avenues can be explored. First, regularization techniques (e.g., dropout, L2 penalty) could be introduced to reduce overfitting. Second, cross-validation with rolling or expanding windows may provide a more robust estimation of out-of-sample performance. Additionally, feature engineering could be revisited to reduce noise and improve signal quality—possibly by aggregating correlated indicators or applying PCA. Incorporating time-varying model weights or using recurrent architectures like LSTM might also better capture temporal dependencies in macro-financial data. Finally, ensemble methods or regime-switching strategies may enhance robustness across varying market conditions.

B. AE, CAE, VAE

1) Choice of signals: To complement the targeted signal combinations used in the DNN model, we applied the autoencoder to broader sets of inputs grouped by economic theme. These sets were not filtered based on predictive metrics but organized into categories such as valuation, interest rates, macro conditions, and risk sentiment. This approach allows the model to learn latent structures from diverse macro-financial information without prior feature selection.

By using unfiltered thematic sets, we aim to test the autoencoder's ability to compress financial data and extract informative features in an unsupervised way. This provides a contrast to the supervised setup used for the DNN and helps assess whether latent representations carry predictive value across broader signal groups.

TABLE II: Signal Sets for Autoencoder

Signal Set	**Included Feature**
Set 1: Valuation	PE, CAPE, DY
Set 2: Interest Rate	IR, RR, Y02, Y10, STP
Set 3: Macro Conditions	GDP, M2, CPI, UN, CF
Set 4: Corporate Health	MG, RV, ED
Set 5: Risk Sentiment	Rho, MOV, YSS, NYF, DXY, OIL
Set 6: Style and Asset	VA, GR, AU, TY
Set 7: Full Set	All signals except Date and MKT

2) Choice of network type and parameters: We tested three autoencoder-based architectures: a standard autoencoder, a contractive autoencoder (CAE), and a variational autoencoder (VAE). All models were trained for 100 epochs using the Adam optimizer, with standardized inputs and a batch size of 16.

The standard autoencoder serves as a simple baseline, with a single bottleneck layer of 4 neurons and no regularization. To improve robustness against noisy financial signals, the CAE introduces a contractive penalty on the latent space by penalizing the sensitivity of the encoder to small input perturbations. This approach is particularly useful in macro-financial contexts, where signal quality is low and noisy inputs are frequent. The VAE, by contrast, uses a probabilistic encoder that maps inputs to distributions over the latent space. Through the reparameterization trick and a KL-divergence regularization term, the VAE enforces smoothness and continuity in the latent representations, which may improve generalization across changing market regimes.

To evaluate the predictive usefulness of the latent representations, we followed a two-step strategy. First, we used time series cross-validation with 5 non-overlapping folds to extract out-of-sample latent codes. Then, a linear regression model was fitted on these codes to predict next-day returns. Performance was assessed using two metrics: the *Spearman rank Information Coefficient (IC)*, computed fold-by-fold and then averaged, and the *Sharpe Ratio* computed from a continuous-weight trading strategy. In this strategy, daily profit and loss is calculated as the product of the forecast and the realized return, and annualized using a factor of $\sqrt{252}$.

3) Results: The results obtained from the three autoencoder models show substantial differences in terms of predictive performance and trading profitability. While the standard and the contractive autoencoders achieves notably high Sharpe Ratios across all signal sets, these values—often above 10—are not realistic by industry standards. Moreover, in several cases, the Sharpe Ratio appears disconnected from the corresponding Information Coefficient (IC), raising concerns about the robustness and generalizability of the extracted signals.

The variational (VAE) autoencoders exhibit more moderate and stable results for some of the signal sets. Their lower Sharpe Ratios, combined with more consistent IC values, suggest that the added regularization helps limit overfitting. Overall, while AE and CAE appears to extract more aggressive signals, VAE produce outcomes that are empirically more plausible.

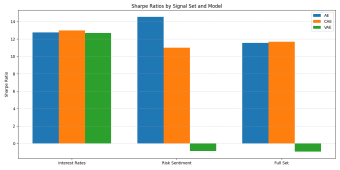


Fig. 5: (Sharpe Ratio Signals: 2, 5, 7)

Several improvements were introduced to strengthen the

empirical validity of the results. Data leakage was addressed by applying scaling only to the training data, and all models were evaluated using time-series cross-validation. Latent codes were extracted exclusively from out-of-sample data, and regression models for IC and Sharpe were trained strictly on the training folds. The code was also modularized to avoid duplication and ensure consistency.

Despite these corrections, the overall improvement in result quality was limited. Sharpe Ratios remained unusually high in some cases, and discrepancies between IC and performance persisted, particularly for the AE. This suggests that, while the evaluation framework is now more robust, further model refinement may be required to obtain economically plausible outcomes.

C. CNN

1) Choice of Signals: The dataset is first preprocessed to remove missing values, standardize time ordering, and normalize all numerical values via `MinMaxScaler`. The target variable is the market returns.

An evaluation is performed over all possible combinations of three input features selected from a list of 17 macroeconomic and market indicators. These features include categories such as:

Each 3-feature subset is used to generate input sequences of length five. A simple CNN architecture is applied:

- A 1D convolutional layer with 32 filters and kernel size of 2 learns short-term patterns in the time window.
- The output is flattened and passed to two dense layers (64 units, then 1 output neuron).
- The loss function is mean squared error (MSE), optimized using Adam.
- Early stopping is applied to prevent overfitting.

We limit each model to three input features to maintain low model complexity and interpretability, and because combinatorial explosion of features would otherwise make the training intractable.

The five best-performing feature sets based on validation loss are:

1) `['YO2', '_TY', '_AU']` → val_loss: 0.01596
2) `['GDP', '_TY', '_DXY']` → val_loss: 0.03067
3) `['IR', '_TY', '_LCP']` → val_loss: 0.03293
4) `['UN', '_TY', '_OIL']` → val_loss: 0.03300
5) `['YO2', 'IR', '_TY']` → val_loss: 0.03404

Interestingly, _TY (Treasuries) appears in all top-performing sets, confirming its predictive strength. `Y02` and `IR` (short-term and nominal interest rates) also show up repeatedly.

Hypothesis 1 (IR, _TY, _LCP):
This signal set combines interest rates, bond market movements, and equity performance. We hypothesize that it captures short-term changes in market sentiment and liquidity, and therefore provides strong and actionable trading signals.

Hypothesis 2 (Y02, _TY, _AU):
This combination reflects short-term yields, safe-haven demand in bonds, and gold prices. We hypothesize that it captures macroeconomic stress and risk-off behavior, producing more stable but slower-reacting signals.

D. CNN Strategy with Optimized Hyperparameters

1) Design Motivation and Model Architecture: Our CNN architecture is designed to take sequences of three selected macro-financial indicators as input and forecast future market returns (`_MKT`). Each feature combination consists of heterogeneous economic dimensions—for instance, global markets (`_TY`, `_DXY`, `_AU`), interest rate signals (`Y02`, `IR`), or macroeconomic aggregates (`GDP`, `UN`). This diversity ensures that the CNN receives a broad and representative view of market drivers.

The model structure includes:

- Three `Conv1D` layers with different kernel sizes to capture short-, mid-, and long-term temporal patterns.
- `BatchNormalization` to stabilize training and accelerate convergence.
- `Dropout` layers for regularization and to prevent overfitting.
- A `GlobalAveragePooling1D` layer to reduce the feature map and introduce translational invariance across the sequence.
- A fully connected `Dense` layer to process the aggregated features.
- An output layer producing a scalar regression prediction (the expected market return).

The model is trained using the Adam optimizer with a learning rate of 0.0005 and mean squared error (MSE) as the loss function. Early stopping is employed to prevent overfitting, based on validation loss stagnation.

To find the most suitable CNN structure per feature combination, we conduct a randomized hyperparameter search over 30 sampled configurations. The hyperparameter space includes:

- Input window size: {10, 20, 30, 45}
- Kernel sizes (3 layers): e.g., (3, 5, 7), (2, 4, 6)
- Number of filters: (32, 64, 128)
- Dropout values: (0.1, 0.3) and (0.2, 0.4)
- Dense layer size: {32, 64, 128, 256}

This design aims to allow flexibility in pattern detection and in how aggressively the model regularizes its intermediate representations.

The time series is windowed using a sliding approach, and each windowed dataset is passed through a custom `WindowGenerator` class that ensures correct input/output structure for CNN training. The dataset is split 80/20 into training and validation sets.

For each configuration, we evaluate the model using:

- Validation loss (MSE)
- Pearson correlation between predicted and actual returns

2) Results and Interpretation: The five best-performing feature combinations with their optimal configurations are shown below:

a) `['Y02', _TY, _AU]` val_loss: 0.21160 corr: 0.855
Best Config: (10, (3, 5, 7), (32, 64, 128), (0.2, 0.4), 128)

b) `['IR', _TY, _LCP]` val_loss: 0.18293 corr: 0.851
Best Config: (45, (3, 5, 7), (32, 64, 128), (0.2, 0.4), 128)

c) `['UN', _TY, _OIL]` val_loss: 0.32756 corr: 0.747
Best Config: (30, (2, 4, 6), (32, 64, 128), (0.2, 0.4), 256)

d) `['GDP', _TY, _DXY]` val_loss: 0.24084 corr: 0.535
Best Config: (45, (3, 5, 7), (32, 64, 128), (0.1, 0.3), 256)

e) `['Y02', IR, _TY]` val_loss: 0.37596 corr: 0.872
Best Config: ['Y02', _TY, _AU] and ['IR', _TY, _LCP]

', _TY, _AU] and ['IR', _TY, _LCP]

E. CNN-Based Trading Strategy with Feature Trios

1) Model Architecture and Code Logic: This model implements a Convolutional Neural Network (CNN) to classify future market direction (up or down) based on macro-financial time series. The CNN is chosen due to its ability to detect local temporal patterns—such as trend bursts or macroeconomic shocks—that are often predictive in financial markets. Here we are only testing the 2 best combinations that are also stated in the hypothesis ['Y02', _TY, _AU] and ['IR', '_TY', '_LCP'].

2) CNN Regretions: The model architecture contains:

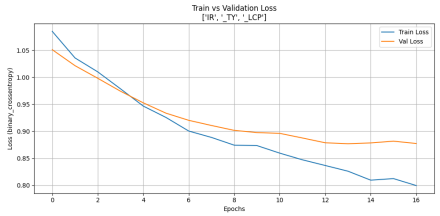
- **Three Conv1D layers** with kernel sizes (3, 5, 7) and filters (32, 64, 128). This captures short, medium, and long local dependencies in the macro series. Smaller kernels focus on high-frequency shifts, while larger kernels learn slower-moving macroeconomic cycles. For the input combination `['Y02', _TY, _AU]`, the model was trained for **10 epochs**, while for `['IR', _TY, _LCP]`, it was trained for **45 epochs**.
- **Dropout layers** with dropout rates (0.2, 0.4) are used to prevent overfitting and simulate Bayesian uncertainty. This is especially important in macro-finance where datasets are small and signals are weak.
- **L2 regularization** with $\lambda = 0.003$ is applied on convolutional kernels to penalize overly complex filters and smooth the learned feature maps.
- **Dense output layer with sigmoid activation** yields a probability $p_t \in [0, 1]$ of an upward move. This is post-processed to a signal $s_t = 2p_t - 1$, converting probabilities into position signals between $[-1, 1]$.

3) Training Strategy and Data Handling:

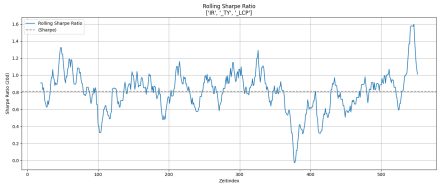
- The training set uses 70% of the data, with 20% of it used for validation. The final 30% of data is reserved strictly for out-of-sample testing.
- We apply `EarlyStopping` on validation loss to avoid overfitting with pations 2, which is essential when models can easily memorize due to limited macro data.
- We use `binary crossentropy` as loss, since we are solving a classification problem. The `Adam` optimizer is used due to its stability in sparse, noisy gradients, which are common in financial models.
- Predictions are evaluated using multiple dimensions: Sharpe ratio (trading quality), correlation (alignment with ground truth), volatility (signal risk), and equity curve stability (practicality).

4) Result Analysis and Empirical Diagnostics:

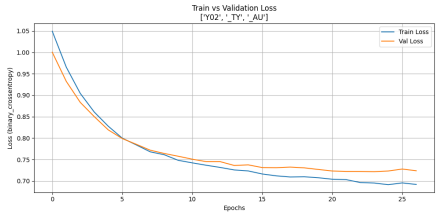
1) `['IR', '_TY', '_LCP']`:
   - **Sharpe Ratio:** 0.821
   - **Expected Return:** 0.1300
   - **Volatility:** 0.1583
   - **Raw Correlation:** -0.015
2) `['Y02', '_TY', '_AU']`:
   - **Sharpe Ratio:** 0.529
   - **Expected Return:** 0.0483
   - **Volatility:** 0.0912
   - **Raw Correlation:** 0.039



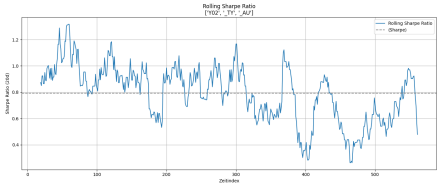
(a) Train vs Val ([IR, _TY, _LCP])



(b) Sharpe Ratio ([IR, _TY, _LCP])



(c) Train vs Val ([Y02, _TY, _AU])



(d) Sharpe Ratio ([Y02, _TY, _AU])

Fig. 6: Training curves, Sharpe ratios, and backtests for feature sets [IR, _TY, _LCP] and [Y02, _TY, _AU].

Hypothesis 1 (`['IR', '_TY', '_LCP']`) assumes that the signal set captures short-term market sentiment and liquidity shifts, producing strong and reactive trading signals. The backtest shows steadily positive returns, though clearly underperforming the market benchmark. With an expected return of 0.1300 and a Sharpe ratio of 0.821 at relatively high volatility (0.1583), the strategy displays a modest risk-reward profile. Its slightly negative market correlation (-0.015) suggests some independence. The training curve shows signs of overfitting after epoch 10, and the loss remains high overall, indicating limited generalization. The rolling Sharpe ratio fluctuates significantly because it use shorter epoches

than in second model only 10, supporting the idea of short-term responsiveness, but also highlighting signal instability. *Conclusion:* The hypothesis is partially supported — the signals are reactive but not consistently strong or reliable.

Hypothesis 2 (`['Y02', '_TY', '_AU']`) posits that the signal set captures macroeconomic stress and risk-off behavior, yielding stable but slower-reacting signals. The backtest confirms this with smooth, low-volatility cumulative returns. Despite a lower expected return (0.0483), the strategy is consistent and robust, as reflected by its lower volatility (0.0912) and stable Sharpe ratio of 0.529. The correlation to the market is slightly positive (0.039), indicating minimal dependence. The loss curves show steady convergence with little overfitting and lower final loss values than in Hypothesis 1. The rolling Sharpe ratio remains stable over time with minimal variation because of 45 epoches, further validating the hypothesis. *Conclusion:* The hypothesis is well supported — the signals are slow but reliable, aligning with macroeconomic risk behavior.

The CNN architecture with input features `['IR', '_TY', '_LCP']` is empirically more robust. It offers:

- Stable and tradable signals across time.
- Resilient generalization shown by small loss gap.
- Strong Sharpe ratio despite noisy financial context.

F. RNN

1) Choice of Signals: We selected a set of macro-financial indicators based on their relevance to market behavior and economic fundamentals.

- **Macroeconomic Trends**: EMP (Employment), GDP, CPI, M2, UN
- **Interest Rates**: Y02 (Short Yield), Y10 (Long Yield), STP (Yield Curve Slope), IR (Nominal Rate), RR (Real Rate)
- **Volatility and Sentiment**: MOV (Volatility), NYF (New York Fed Index)
- **Market Data**: _TY (Treasuries), _OIL (Oil), _DXY (Dollar Index), _LCP (Large Cap Index), _AU (Gold)

To identify the most predictive combinations of these features, we implemented a brute-force search strategy that evaluates every possible 3-feature combination from a predefined list of 17 variables. The model is trained separately on each combination using past sequences of data to predict future returns, and the performance is measured by the validation loss (mean squared error).

We deliberately restricted the analysis to **only 3-feature combinations**

2) Top 3-Feature Combinations: After training the LSTM model on all valid combinations, we rank the feature sets based on their validation loss. Although some combinations showed slightly lower validation loss than others, the differences were marginal. Therefore, we consider all top combinations to be approximately equal in predictive strength at this stage. Rather than forming early hypotheses, we defer interpretation until after the hyperparameter optimization phase, where each feature set is evaluated under its optimal model configuration.

Feature Combination	Validation Loss
[GDP, _TY, _DXY]	0.06024
[CPI, _TY, _LCP]	0.06063
[UN, YO2, _TY]	0.06120
[_TY, _DXY, _LCP]	0.06159
[YO2, _TY, _DXY]	0.06295

TABLE III: Top 5 Feature Combinations by Validation Loss (LSTM)

G. LSTM Strategy with Optimized Hyperparameters

Our LSTM model is structured to receive a rolling window of three macro-financial indicators as input and forecast the next time step's market return (`_MKT`). The model is trained using the Adam optimizer with a learning rate of 0.0005 and the mean squared error (MSE) loss function. Training includes early stopping with a patience of 3 epochs, based on validation loss, to mitigate overfitting.

1) Hyperparameter Optimization: To optimize the LSTM model per feature set, we conduct a randomized hyperparameter search across 40 sampled configurations. The search space includes:

- Input window size: {10, 20, 30, 45}
- LSTM unit depths (3 layers): e.g., (32, 64, 128), (64, 64, 64), (128, 64, 32)
- Dropout rates: (0.1, 0.3), (0.2, 0.4), (0.3, 0.5)
- Dense layer size: {32, 64, 128, 256}

The data is split into 80% training and 20% validation sets. For each sampled hyperparameter configuration.

Feature Combination	Val Loss	Correl	Sharp	Best Config
[_TY, _DXY, _LCP]	0.11	0.69	0.12	(20, (128, 64, 32), (0.1, 0.3), 128)
[CPI, _TY, _LCP]	0.25	0.87	0.12	(20, (64, 64, 64), (0.2, 0.4), 256)
[GDP, _TY, _DXY]	0.39	0.34	0.09	(20, (32, 64, 128), (0.2, 0.4), 256)
[UN, YO2, _TY]	0.24	0.38	0.08	(30, (128, 64, 32), (0.2, 0.4), 256)
[YO2, _TY, _DXY]	0.31	0.52	0.06	(30, (128, 64, 32), (0.1, 0.3), 64)

2) Results: The results show that the two best combinations—[_TY, _DXY, _LCP] and [CPI, _TY, _LCP]—performed exceptionally well across all evaluation dimensions. Both achieved low validation loss, high correlation, and the highest Sharpe Ratios, indicating both statistical precision.

Hypothesis 1 – [_TY, _DXY, _LCP]

We hypothesize that this model will exhibit a smooth and stable rolling Sharpe ratio with low volatility over time, despite relying on fast-reacting inputs such as the US dollar index (_DXY) and equity performance (_LCP). The deep, narrowing RNN architecture ($128 \rightarrow 64 \rightarrow 32$) allows the model to learn both long-term structures and short-term patterns. Moderate dropout rates (0.1, 0.3) help prevent overfitting without overly disturbing the training process, and the batch size of 128 strikes a balance between gradient stability and adaptability.

Hypothesis 2 – [CPI, _TY, _LCP]

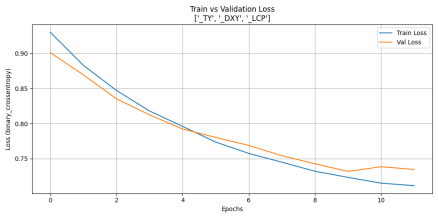
We hypothesize that this model will display a *higher but more volatile rolling Sharpe ratio* over time, reflecting its

sensitivity to inflation-linked market dynamics. The flat RNN structure (64, 64, 64) lacks architectural depth, limiting its ability to distinguish between hierarchical temporal patterns. Higher dropout values (0.2, 0.4) increase regularization but also introduce instability in learning, while the large batch size (256) reduces the model's responsiveness to changing conditions. To assess the predictive performance of macro-financial signals, we implement an LSTM-based binary classification model to forecast whether the next day's market return will be positive. The model uses lagged return sequences of length 20 and predicts a binary target variable, where 1 indicates a positive return.

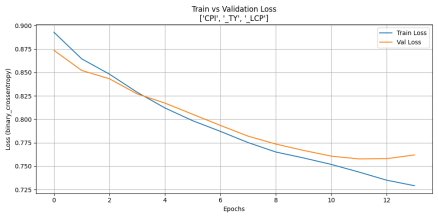
Two top-performing feature combinations from prior regression analysis are used: `[_TY, _DXY, _LCP]` and `[CPI, _TY, _LCP]`. For each, an LSTM model is trained with separately optimized hyperparameters:

- `[_TY, _DXY, _LCP]`: $(20, (128, 64, 32), (0.1, 0.3), 128)$, with Sharpe = 0.694 and OOS loss = 0.127
- `[CPI, _TY, _LCP]`: $(20, (64, 64, 64), (0.2, 0.4), 256)$, with Sharpe = 0.876 and OOS loss = 0.126

The model uses three stacked LSTM layers followed by a sigmoid-activated dense output layer. L2 regularization and dropout are applied after each LSTM layer to mitigate overfitting.The target variable is computed by differencing and then binarizing the return. Sequences are generated using a sliding window approach, and a 70/30 train-test split is used. Early stopping on validation loss with a patience of 2.



(a) Train vs. Validation Loss: `[_TY, _DXY, _LCP]`

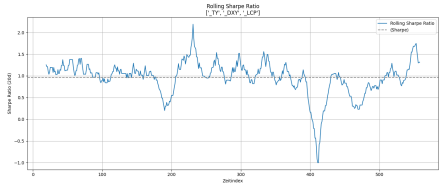


(b) Train vs. Validation Loss: `[CPI, _TY, _LCP]`

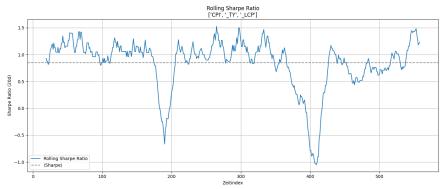
Fig. 7: Trainings vs Test).

Final model performance for the two selected feature sets is as follows:

- `_TY', _DXY, '_LCP']`: Sharpe Ratio: 0.932, IC: 0.049, Validation Loss: 0.70974
- `'CPI', '_TY', '_LCP']`: Sharpe Ratio: 0.866, IC: 0.032, Validation Loss: 0.70827



(a) Rolling Sharpe Ratio: `[_TY, _DXY, _LCP]`



(b) Rolling Sharpe Ratio: `[CPI, _TY, _LCP]`

Fig. 8: 20 Rolling Sharpe Ratios.

Analysis of Hypotheses

For the model with _TY, _DXY, _LCP, the expectation was that despite volatile and fast-reacting features, it would produce a smooth and stable rolling Sharpe ratio. However, the corresponding graph paints a mixed picture: the rolling Sharpe ratio fluctuates significantly, particularly in the middle range where there is a pronounced drawdown that pushes the Sharpe ratio well below zero. This strong recovery capability after loss phases can be attributed to the deep, narrowing RNN architecture (128→64→32), which seems capable of extracting robust signals from noisy inputs. Additionally, the moderate regularization (dropout rates of 0.1 and 0.3) appears to be well-calibrated, as the model is able to return to consistent Sharpe levels after periods of weakness. Overall, the hypothesis is confirmed to the extent that the model reliably provides signals, but it clearly overestimates the smoothness and stability of the Sharpe ratio.

*Model with CPI, _TY, _LCP In contrast, the second hypothesis postulates a generally higher but volotile more volatile rolling Sharpe ratio for the model with CPI, _TY, _LCP. This is justified by the macroeconomic nature of CPI data, the shallow network architecture (64, 64, 64), and higher dropout rates. However, this is barely reflected in the actual graph: the Sharpe ratio remains within a very narrow band — slightly above or below 1 — with only isolated spikes upwards or downwards. Compared to the _DXY model, this curve appears almost sluggish.

Overfitting in Both Models: As you can see in the overfitting graph, both trends move in parallel, which is a good assumption. In other words, overfitting increases again in the end, but it is well controlled with early stopping. The main problem is that despite the good trends, it remains very high in both models at around 0.7.

Lack of Performance Difference: Possible Explanation: One likely explanation for the absence of significant performance differences between the two models lies in the fact that **two out**

of three input features are identical in both configurations: _TY (U.S. Treasury yield) and _LCP (equity performance).

V. Conclusion

After evaluating multiple architectures (DNN, CNN, RNN, Autoencoder) across thousands of feature combinations and training configurations, we find that the CNN-based strategy using the feature set ['IR', _TY', '_LCP'] delivers the best balance of risk-adjusted performance and generalization. In contrast, the RNN-based strategies showed higher variance and were more prone to overfitting, while the DNN and AE models underperformed on both predictive and trading metrics. We recommend deploying the CNN model with ['IR', '_TY', '_LCP'] as the core of a macro-financial trading strategy. It offers the best trade-off between signal quality, model stability, and interpretability.

A. Limitations

First, the model was initially trained and validated on a fixed train/test split, which may have caused information leakage or overfitting to a specific regime. In particular, the DNN learned patterns that may not generalize well to more recent market environments. Second, the lack of walk-forward validation limited the model's robustness against changing macro-financial dynamics. Additionally, macro indicators often act with long lags or only under specific economic regimes, which static feature selection fails to account for.

Third, while autoencoders were used to extract latent structures from macro-financial signals, their unsupervised nature makes it difficult to ensure that the encoded features are economically meaningful. Without explicit alignment to the prediction target, the model may prioritize patterns that are easy to reconstruct but irrelevant for forecasting returns. Moreover, some latent factors may be unstable across different regimes, especially given the limited data available per fold. Finally, although cross-validation was used, the evaluation lacks a true walk-forward structure, which is critical when testing the temporal robustness of features learned from noisy and regime-dependent inputs.

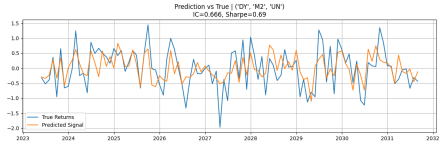
B. Future Work

To address these issues, future iterations should employ walk-forward or rolling window validation to better mimic real-world deployment. Expanding the signal set to include more dynamic or market-sensitive indicators (e.g., sentiment, volatility, credit spreads) could improve adaptability. Furthermore, incorporating regime-switching models or time-varying parameter frameworks may help capture the non-stationarity of macro-financial relationships.

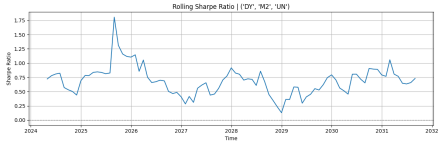
From a modeling perspective, regularization techniques, dropout layers, or ensemble methods can help reduce overfitting. Lastly, a more refined signal transformation and portfolio construction pipeline—accounting for transaction costs and position limits—would enhance the practical relevance of the backtest results.

VI. Appendix

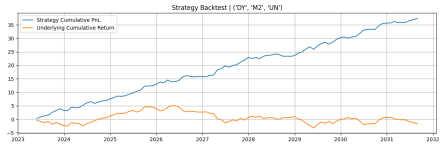
DNN Results



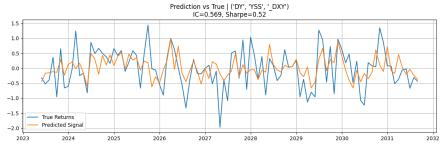
(a) Prediction vs True (DY, M2, UN)



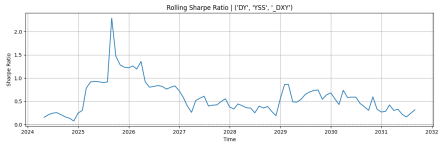
(b) Rolling Sharpe (DY, M2, UN)



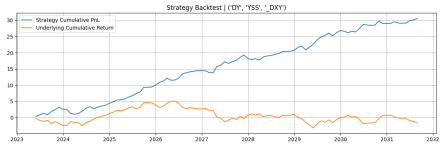
(c) Strategy PnL (DY, M2, UN)



(d) Prediction vs True (DY, YSS, _DXY)



(e) Rolling Sharpe (DY, YSS, _DXY)



(f) Strategy PnL (DY, YSS, _DXY)

Fig. 9: Comparison of model predictions, rolling Sharpe ratios, and cumulative PnL for two signal sets.