Learning from memory: Asset Pricing via Recurrent Neural Network and Attention Mechanism*

Yilun Wang[1,a,*]

[a]*North Carolina State University, Department of Economics, Raleigh, U.S.*

Abstract

We develop a novel nonlinear conditional asset pricing framework that employs recurrent sequence models with attention to capture the temporal evolution of factor loadings. By jointly modeling long and short horizon dependencies, the architecture uncovers predictive structure that traditional linear specifications and machine learning methods systematically miss. Using more than 40 years of U.S. equity data, we show that sequence based latent factor model deliver markedly higher out-of-sample R^2, Sharpe ratios, and cost-adjusted portfolio performance relative to standard econometric and machine learning benchmarks. The results demonstrate that attention augmented sequence learning provides a scalable and economically meaningful advancement for forecasting returns and modeling dynamic characteristic return relationships.

Keywords: Latent factor model, Deep Learning, Recurrent Neural networks, Attention Mechanism, Return predictability

1. Introduction

Forecasting financial market dynamics, particularly stock market returns, is a central theme in economic research. Accurate forecasts can guide profitable decisions, like portfolio construction and stock investments. Tradi-

*We thank Mehmet Caner, Ilze Kalnina, Zheng Li, Denis Pelletier, Andreas Neuhierl, participants at the 2024 Chinese Economists Society (CES) China Conference, and Asia Meeting of the Econometric Society, East & Southeast Asia conference for helpful comments. All errors and omissions are ours.

*Corresponding author

Email address: ywang323@ncsu.edu (Yilun Wang)

tional asset pricing models, while fundamental, often struggle with high-dimensional data, nonlinear market relationships, and low signal-to-noise ratios in financial data. Machine learning, especially Deep Learning, emerges as a potent alternative, adept at processing large datasets and modeling complex nonlinearities. Deep Learning models surpass traditional methods by capturing nonlinearities and interactions in a data-driven way, offering more precise return predictions, a significant advancement in the field of economic forecasting. (Chinco et al., 2019; Gu et al., 2020; Callot et al., 2021; Caner et al., 2023; Chen et al., 2023; Nagel, 2021)

Ross (1976) 's arbitrage pricing theory posits that expected returns are offset by factor exposures. In his static asset pricing model, factors are known and observed. However, traditional static model meets huge challenges like time-varying betas, which means that factor loadings are likely to change over time as firms evolve. (Bicksler and Jensen, 1973; Ghysels, 1998; Gagliardini et al., 2016; Büchner and Kelly, 2022; Kelly et al., 2023) The inability to capture return dynamics spurred the exploration of latent factor models, in which the situation beta is dynamic and time-varying. (Connor and Korajczyk, 1986; He et al., 2017; Ang et al., 2020; Kozak et al., 2020) The form of the Latent factor model is:

$$r_{i,t} = \beta\left(z_{i,t-1}\right)' f_t + u_{i,t} \tag{1}$$

where factor loadings $\beta(z_{i,t-1})'$, which has K dimensions, is a nonlinear function with observed asset characteristics $z_{i,t-1}$, which has P dimensions. Kelly et al. (2019) propose a novel method named the instrumented principal component analysis (IPCA) to estimate factor exposures. In this method, the firm characteristics, $z_{i,t-1}$, are used as instrumental variables to identify latent factors and calculate a linear beta function. The authors have discovered that asset characteristics serve as valuable predictors of returns by accurately identifying compensated exposures to overall risk. Consequently, this feature makes them useful in forecasting anomalies.

The linear assumption for factor loadings β based on characteristics data $z_{i,t-1}$ is convenient and has been employed by traditional asset pricing theory. Fan and Mincheva (2011); Fan et al. (2013) applied both observable and latent linear factors to a vast array of assets. They demonstrated that it's possible to reliably determine the precision matrix for the returns of assets for high dimensions by integrating linear factor models with a sparse error covariance matrix. Caner et al. (2023) proposes a residual based nodewise

regression on factor model to estimate the precision matrix of error and returns, and they also get maximum Sharpe Ratio consistency. However, linear approximation for risk exposures based on observable characteristics data has been challenged. Many theoretical and empirical asset pricing models predict nonlinearities in return dynamics as a function of state variables. (Campbell and Cochrane, 1999; Bansal and Yaron, 2004; Menzly et al., 2004; He et al., 2013).

To solve the nonlinearity, Fan et al. (2015) replace the assumption that factor betas are linear in characteristics with an assumption that factor betas are nonparametric functions of characteristics. Freyberger et al. (2020) explore the use of a non-linear additive approach to account for asset returns. They suggest a flexible group lasso method and demonstrate through a practical U.S. stock market data analysis that a portfolio created using a non-linear factor model can yield a much more favorable return-to-risk ratio than one derived from a linear factor model. More deep learning skills have recently been employed to solve the non-linear relationship. Caner and Daniele (2023) develop a deep neural network model to get a consistent estimator of the precision matrix of asset returns in large portfolios based on deep learning residuals formed from non-linear factor models, and the model works well even in low signal-to-noise environment. Gu et al. (2021) introduce the Conditional Autoencoder (CAE), a deep learning-based asset pricing model that allows both factors f_t and risk exposures $\beta(z_{i,t-1})$ to be entirely data-driven and time-varying. They construct f_t using an autoencoder, where an encoder extracts latent factors from observed excess returns, and a decoder reconstructs the returns, ensuring that the learned factors capture key pricing information. Meanwhile, $\beta(z_{i,t-1})$ is modeled through a neural network that maps asset characteristics to risk loadings, allowing for nonlinear and dynamic interactions between firm-specific features and factor exposures. This flexible framework improves predictive accuracy and cross-sectional pricing performance over traditional models like IPCA. Our work builds upon their framework by continuing their $\beta(z_{i,t-1})$ network setting but modifying the f_t network. Specifically, we introduce temporal dependencies into the f_t network to better capture the dynamic evolution of latent factors over time. Unlike Gu et al. (2021), where f_t is independently learned at each time step through an autoencoder, we incorporate a recurrent structure that allows past factors to influence current factor estimates. By doing so, our model leverages sequential patterns in asset returns, and strengthens predictive ability, particularly in settings where market conditions evolve

dynamically.

Most deep learning research does not account for temporal dependencies in time series data, where current values often hinge on past data. This is particularly true for financial data like asset prices, influenced by historical trends and external factors. Ignoring these dependencies can lead to inaccurate predictions. Traditional Deep Neural Networks (DNNs), consisting of multi-layered neurons processing data independently, lack mechanisms to handle such temporal sequences, treating input features as unrelated. To address this, Recurrent Neural Networks (RNNs) have been developed. Unlike DNNs, RNNs possess internal states that capture previous input information, passing it across time steps to create a form of memory. This allows them to recognize the order and dependencies of data points. RNNs have gained prominence in areas like Natural Language Processing (Sutskever et al., 2014; Hewamalage et al., 2021) and have been increasingly applied to time series forecasting (Moghar and Hamiche, 2020; Sako et al., 2022), demonstrating their capability to explore temporal relationships.

The first contribution of our work is integrating a Recurrent Neural Network (RNN) structure into the latent factor model to capture temporal dependencies in returns. Some scholars, such as Chen et al. (2023), utilize an RNN to estimate a small number of macroeconomic states, which are then combined with firm characteristics in a deep learning framework based on Generative Adversarial Networks (GAN) to extract nonlinear factor structures in a cross-sectional setting. However, their model does not explicitly account for temporal dependencies in factor evolution. In contrast, our approach directly models the evolution of latent factors over time. By incorporating an RNN, we ensure that past factors influence current estimates, capturing dynamic patterns in systematic risk that purely cross-sectional models overlook. Unlike traditional feedforward neural networks that treat sequence elements (like a series of returns) independently, lacking memory of past inputs and understanding of data point order and dependencies, our model processes sequences element by element while maintaining a “ hidden state" that updates at each step. “ Hidden state" acts as a memory, enabling the network to “remember" previous sequence elements and use this information to shape predictions. The output is from both current inputs and this memory. This capability allows our model to detect temporal dependencies by adjusting the weights between hidden states and current inputs. Our approach goes beyond just processing time series inputs, as it also considers the sequence order by retaining past information, setting it apart from

traditional neural networks.

Our second contribution is a computational improvement that enhances the training efficiency of deep learning-based models. We propose an attention-enhanced structure to mitigate the vanishing gradient problem, which arises when training deep networks on long-horizon asset pricing data. Unlike standard deep learning-based approaches, our model integrates an attention mechanism (Vaswani et al., 2017) to dynamically reweight feature importance at different time steps, effectively shortening the gradient path. This modification allows gradients to propagate more efficiently through deeper layers. By introducing attention into factor models, we enhance computational efficiency of extracted factors, offering a solution for long-horizon asset pricing datasets.

Since the early development of deep neural networks, the vanishing gradient problem has been a persistent challenge in training deep architectures (Hochreiter, 1991; Bengio et al., 1994). When backpropagating gradients through many layers, small gradients exponentially shrink, leading to slow convergence and ineffective learning in early layers (Pascanu et al., 2013). While techniques such as Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) have been introduced to mitigate this issue, it remains a fundamental problem when training deep models on datasets with long time horizons.

In our study, we utilize an asset pricing dataset covering more than 40 years of historical data, making the vanishing gradient problem particularly pronounced during training. We encountered severe training inefficiencies due to gradient decay when learning long-horizon latent factors, leading to slow convergence and suboptimal factor representations. By using the attention mechanism, our model adaptively assigns different importance weights to historical return sequences, allowing it to retain relevant long-term dependencies while filtering out less informative signals. Empirical results demonstrate that our method achieves a higher R^2 and Sharpe ratio compared to linear models and traditional machine learning models, highlighting its improved predictive ability.

In practice, attention-based architectures have demonstrated remarkable success in handling long-term dependencies across various domains, such as speech recognition (Subakan et al., 2021), image classification (Zhang et al., 2019), and text generation(Kitaev et al., 2020). By applying these insights to asset pricing, our model ensures more stable training, improves factor interpretability, and enables deep learning techniques to scale efficiently to

long-horizon financial datasets.

Our model was applied to analyze 44 years of monthly U.S. equity returns, and results indicate that it significantly surpasses benchmark models in various out-of-sample evaluations. Statistically, our model achieves superior out-of-sample total and predictive R^2 for individual returns at 42.65% and 6.17%, respectively, outperforming all benchmark models. Economically, it delivers the highest annualized Sharpe ratios of 3.03 for equal-weighted long-short portfolios and 2.39 for long-only portfolios without transcation cost. Besides, our model exhibits greater stability over benchmark models in out-of-sample performance, concerning the number of latent factors, K.

Finally, our study adds to the extensive and effective use of machine learning and deep learning within the financial sector. Heaton et al. (2017) use deep learning hierarchical models for financial prediction and classification problems. They find that deep learning can detect and exploit interactions invisible to existing economic theory. Gu et al. (2020) provide a comprehensive empirical investigation of return prediction performance using firm characteristics by multiple machine learning algorithms. They show that trees and neural networks are the best performing methods than traditional statistical models. Ke et al. (2019) introduce a new text-mining methodology to extract information from news articles to predict asset returns and Jiang et al. (2023) use convolutional neural network (CNN) with image data to predict stock returns. Bianchi et al. (2021) show that trees and neural networks (NNs) methods provide strong statistical and empirical evidence in treasury bond return prediction. Bali et al. (2020) extend the results into corporate bonds. Feng and He (2022) consider the Bayesian Hierarchical modeling to estimate the expected returns and residual covariance matrix. Bryzgalova et al. (2020) develop regularized portfolios using basis portfolios generated by the decision tree, while Xuanling Yang and Zhu (2023) employ variational autoencoder to calculate the mean and variance of expected returns, respectively.

The rest of the paper is organized as follows. In Section 2, we introduce our methodology and show how we combine RNN and Attention Mechanism to construct our model. Section 3 presents our empirical studies. Section 4 concludes.

2. Methodology

In this section, we begin by introducing the Recurrent Neural Network (RNN) and differentiating it from the traditional feed-forward networks (FFNs). Next, we demonstrate the process of dimension reduction with RNN structure. We utilize an encoder-decoder structure combined with RNN to handle long temporal dependency data. Finally, we will construct and evaluate our model, which is designed to better explore time dependencies in sequence data.

2.1. Recurrent Neural Network(RNN)

2.1.1. RNN Structure

RNN is a class of neural networks designed for processing sequential data, capturing dynamic temporal behavior. The fundamental feature of RNN is its internal hidden state, or memory, which captures information about previous inputs. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike FFN, RNN can use its internal state to process sequences of varying lengths.

The output $y_t \in \mathbb{R}^K$ of RNN structure with L hidden layers can be written as follows:

$$h_t^1 = f\left(W_{hh}^1 h_{t-1}^1 + W_{xh}^1 x_t + b_h^1\right) \tag{2}$$

$$h_t^l = f\left(W_{hh}^l h_{t-1}^l + W_{xh}^l h_t^{l-1} + b_h^l\right) \tag{3}$$

$$y_t = W_{hy}^L h_t^L + b_y^L \tag{4}$$

where $l = 2, \ldots, L$ denotes the number of layers. $h_t^1 \in \mathbb{R}^{n_h^1}$ is the time t step hidden state in the first hidden layer, $h_t^l \in \mathbb{R}^{n_h^l}$ is the time t step hidden state in the layer l, $h_{t-1}^l \in \mathbb{R}^{n_h^l}$ is the time $(t-1)$ step hidden state in the layer l, $x_t \in \mathbb{R}^N$ is the current input. We will use h_0^1, which is a zero vector to initialize the hidden state. From the second layer, the output of the previous layer h_t^{l-1} will replace x_t to be the new current input, $W_{hh}^l \in \mathbb{R}^{n_h^l \times n_h^l}$ is the weights for the hidden state from previous information, or memory. $W_{xh}^l \in \mathbb{R}^{n_h^l \times n_h^{l-1}}$ is the weights for the current inputs. $f(\cdot)$ is the activation function.

The process of computing the output y_t in an RNN with L hidden layers begins with the first equation (2), where the hidden state of the first layer h_t^1 at time step t is computed using a nonlinear activation function $f(\cdot)$ applied to the weighted sum of the previous hidden state h_{t-1}^1, the current input x_t,

and a bias term. For subsequent layers ($l = 2, \ldots, L$) in (3), each hidden state h_t^l is computed based on three components: the hidden state from the previous time step in the same layer (h_{t-1}^l), capturing temporal dependencies, and the hidden state from the previous layer at the current time step (h_t^{l-1}), ensuring hierarchical information flow, and a bias term. The final equation (4) determines the output y_t by applying a linear transformation (a weighted sum with bias) to the topmost hidden layer's state h_t^L. This structure allows the RNN to model sequential dependencies across time while also enabling deeper representations through multiple hidden layers.

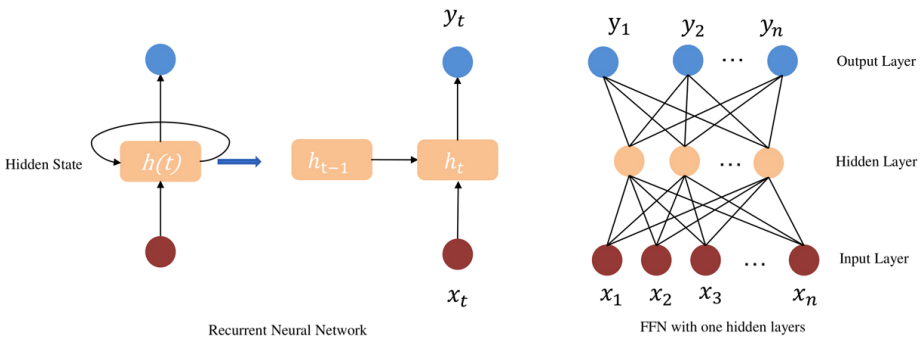

Figure 1: RNN and FFN structure.

Figure 1 shows the RNN and FFN structure with one layer. Left panel describes recurrent neural network, which shows that current output y_t is learnt from the hidden state h_t, and h_t is updated from the previous memory h_{t-1} and current input x_t. The right figure is an example of one hidden layer feedforward neural network.

2.1.2. Sequence Model

We can use RNN to build a Sequence model, which can convert one sequence into another.[1] At its core, a Sequence model is similar to a standard autoencoder structure, which comprises two main components: the encoder and the decoder. The encoder processes the input sequence x_t, which is $N \times 1$ vector, and compresses the information into a hidden fixed-size context vector

[1]The first sequence model is created by Sutskever et al. (2014) and Cho et al. (2014) for machine translation. In their works, they come up with a structure with two RNNs: One RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols.

C, which is 1×1. This context vector C, which aims to capture the essence of x_t, is then fed into the decoder, which generates the output sequence f_t, which is a $K \times 1$ vector, step by step. The model is trained to maximize the likelihood of the correct output sequence f_t given the input sequence x_t. Figure 2 compares the architecture of Sequence model and standard autoencoder.

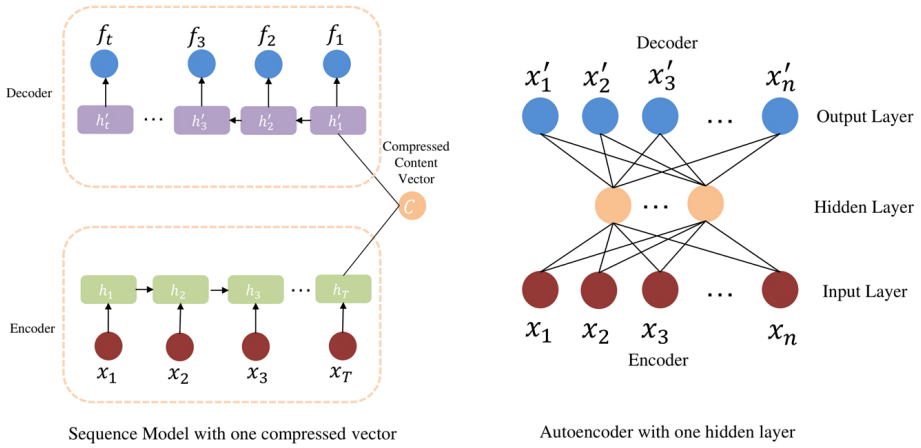

Figure 2: The architecture of the Sequence Model and standard autoencoder model.

The encoder takes the input sequence x_t, one token at a time, to update its hidden state h_t, which includes the previous information. The encoder's final hidden state h_T is then passed as the context vector C to the decoder. The decoder uses each decoder hidden state h'_t to generate the output sequence f_t. Each time step generates a probability distribution over the possible next tokens. The token with the highest probability is then chosen as the output, and the process continues until the end of the output sequence f_t is reached.

Figure 2 shows that both the Sequence model and Autoencoder compress high-dimensional inputs into a low-dimensional hidden layer and then expand to the output. In the Sequence model, the decoder predicts the target sequence token by token using the compressed vector, while the Autoencoder uses hidden layer neurons to represent latent variables for compressed input representation.

2.2. Challenge of RNN and solutions

2.2.1. Vanishing Gradients

Like FFN, RNN also needs to calculate the gradients for weight matric and bias terms for estimation and then update them with a learning rate α. However, during the process, RNN is often challenged by vanishing gradients problems during training. This limitation hinders their ability to learn from data points that are far apart in time.

Vanishing gradients[2] occur when gradients, which are used to update network weights through backpropagation, become extremely small, effectively preventing the network from further learning or making the learning process very slow. This problem arises because of the multiplication of gradients through layers. As the number of layers increases, the gradients of the network's previous layers tend to decrease exponentially, leading to minimal adjustments in these layers during training. This phenomenon makes models difficult to learn long-range dependencies or complex patterns in data.

2.2.2. Solutions to Vanishing Gradients

Long Short-Term Memory network (LSTM) and Gated Recurrent Unit (GRU) are two special RNN structures, which are designed to avoid the vanishing gradient problem. They contain structures called gates to regulate the flow of previous information and output hidden states. These gates allow the network to maintain long-term dependencies and learn which data to store in long-term memory and which data to discard. LSTM has three gates: the input, forget, and output gates, and one cell state, which is used to control the long-term memory, while GRU is a simplified version of LSTM. It has only two gates: the update and reset gate.The structures of LSTM and GRU structure have been shown in the Appendix C. We use both LSTM and GRU in our sequence models to avoid vanishing gradients.

Besides, in order to capture long-range dependencies in sequence data, Bahdanau et al. (2014) introduced a neural network architecture, attention mechanism, which can build a series of unique context vectors C_i for every decoder time-step based on different weighted aggregations across all of the encoder hidden states h_t. It allows the model to "focus" on different parts of the input sequence when producing each token of the output sequence. Instead of relying solely on one context vector C, the attention mechanism

[2]We add details about vanishing gradients in Appendix B

computes a weighted sum of all encoder hidden states h_t based on their relevance to the current decoding step. These weights are assigned by the mechanism, identifying relevant encoder states for each decoder state.

During decoding, each step involves calculating an alignment score e between the decoder's previous hidden state h'_t and each encoder hidden state h_t. This score assesses the relevance of each input token to the current output token. The scores undergo normalization to generate attention weights α, which dictate the decoder's focus on each encoder state. These weights then create a weighted sum of encoder states, forming context vectors C_i that encapsulate relevant input sequence information for each step. The decoder combines these context vectors with its current state to generate the output token. Figure 3 illustrates this attention mechanism.

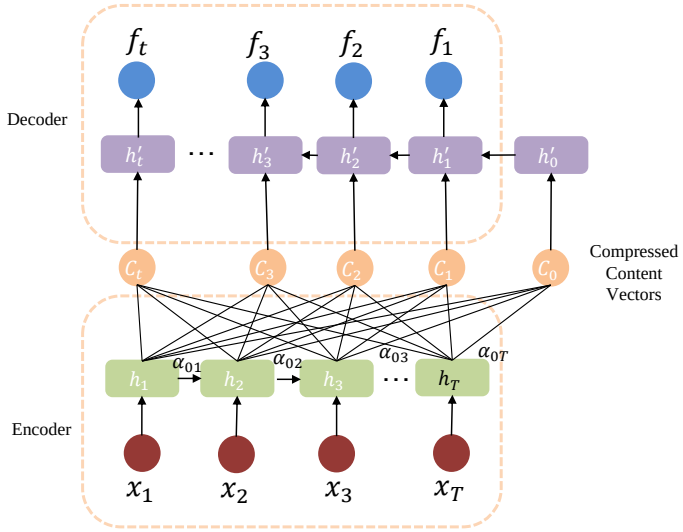

Figure 3: The architecture of attention mechanism in Sequence model

2.3. Model Description

Our network model is inspired by the framework from Gu et al. (2021), which consists of two key components: factor network and beta network. The primary improvement of our model lies in the formulation of the factor network. We learn a compressed content vector C_i from individual returns r_t $(N \times 1)$ and their hidden states h_t, and then reconstruct r_t from the compressed representations C_i with minimal loss of information to generate the

output factor f_t $(K \times 1)$. Unlike Gu et al. (2021), who estimate latent factors using a conditional autoencoder without explicitly modeling temporal dependencies, our model introduces a recurrent structure that ensures f_t depends not only on current inputs but also on past states, thereby capturing long-term dependencies in asset returns. For beta network, we follow Gu et al. (2021)'s settings to apply deep neural network to explore non-linear relation between firm characters and beta. Then we combine factor network and beta network to build our model based on both sequence model and attention mechanism.

2.3.1. Factor Network: Sequence Model with Attention Mechanism

We construct our factor network part with Attention-Mechanism because this method can provide a way to dynamically weigh the importance of different parts of the input sequence during the decoding process and solve traditional RNN problems like vanishing gradient.

We start from the encoder with Attention-Mechanism. The first step is to give an initial value to the encoder hidden state.

$$h_0 = 0 \tag{5}$$

where h_0 is a zero vector of shape $d_h \times 1$. It is the initial hidden state from encoder, d_h is the number of hidden units of encoder hidden states. Next, at each time step t, the encoder updates its hidden state.

$$h_t = g_1(W_1 r_t + W_2 h_{t-1} + b_1) \tag{6}$$

where h_t $(d_h \times 1)$ is the vector of encoder hidden state, r_t $(N \times 1)$ is a vector of the individual returns, W_1 $(d_h \times N)$ and W_2 $(d_h \times d_h)$ are learned weights parameters that are shared across all time steps. b_1 $(d_h \times 1)$ is the vector of bias parameters. g_1 is an activation function.

After we get all encoder hidden states, we use the last encoder hidden state h_T to get the initial hidden state h'_0 for decoder to calculate attention scores.

$$h'_0 = h_T \tag{7}$$

where h'_0 $(d_h \times 1)$ is the initial hidden state for decoder. For each decoder step i, we compute an attention score with each encoder hidden state:

$$e_{i,t} = h'_i W_{\text{att}} h_t \tag{8}$$

where $e_{i,t}$ (1×1) is the attention score between hidden encoder state and decoder state. h_i $(d_h \times 1)$ is decoder hidden state at step i, h_t $(d_h \times 1)$ is encoder hidden state at time t, W_{att} $(d_h \times d_h)$ is the attention weight matrix. The attention score $e_{i,t}$ determines how much attention the decoder should pay to each encoder hidden state at i step.

We normalize the attention scores:

$$\alpha_{i,t} = \frac{\exp(e_{i,t})}{\sum_{t=1}^{T} \exp(e_{i,t})} \tag{9}$$

where $\alpha_{i,t}$ (1×1) is the weight of encoder's hidden state h_t. Then, the compressed feature vector is computed as a weighted sum of encoder hidden states:

$$C_i = \sum_{t=1}^{T} \alpha_{i,t} h_t \tag{10}$$

where C_i $(d_h \times 1)$ is the compressed feature with dynamic weighs through the different encoder hidden state h_t

During encoding step, we first initialize the input sequence with equation (5), then we get the input's "memory", h_t, from the last time step temporal dependency h_{t-1} and current time step input r_t with equation (6). After obtaining encoder hidden states, we could calculate compressed vector in each encoding step with equation (10). Equation (9) and equation (8) shows the process to get attention weights and attention scores, respectively. Through the encoding step, we can obtain a vector C_i, which includes inputs' memory from each time step.

For decoding step, we get f_t by decoding C_i and decoder hidden state h'_t. we show the decoder:

Decoder with Attention-Mechanism:

$$h'_i = g_2(W_3 h'_{i-1} + W_4 C_{i-1} + b_2) \tag{11}$$

where h'_i $(d_h \times 1)$ are decoder hidden states, W_3 $(d_h \times d_h)$ and W_4 $(d_h \times d_h)$ are weights parameters, and b_2 $(d_h \times 1)$ is a vector of bias parameters. Through this step, we can gather all decoder hidden state H

$$H = [h'_1, h'_2, ..., h'_T] \tag{12}$$

The last step is to calculate factors from decoder hidden states H.

$$f_t = W_{\text{factor}} H + b_3 \tag{13}$$

where $f_t(K \times 1)$ are final factor output, K is the number of factors, $W_{\text{factor}} \in \mathbb{R}^{K \times d_h T}$

The equation (11) shows the evolution process of h'_i. The decoder hidden state is from the compressed vector and its last time step memory. The equation (13) shows that factor output f_t is from each period decoder hidden state h'_i. Through the decoding process, we get the expected return $f_t(K \times 1)$ by learning the temporal dependencies.

2.3.2. Beta Network: Feedforward neural network

In the second part, we follow Gu et al. (2020)'s settings and design to use the feedforward structure for beta network. We consider $\beta_{i,t-1}$ as a feedforward neural network of firm characteristics $z_{i,t-1}$, which shows a nonlinear functional form between factor exposures and characteristics.

$$s^0_{i,t-1} = z_{i,t-1} \tag{14}$$

$$s^l_{i,t-1} = g_3\left(W^{l-1} s^{l-1}_{i,t-1} + b^{l-1}\right), \quad l = 1, \ldots, L_\beta \tag{15}$$

$$\beta_{i,t-1} = W^{L_\beta} s^{L_\beta}_{i,t-1} + b^{L_\beta} \tag{16}$$

The first equation (14) initializes the network as a function of the characteristic data $z_{i,t-1}$ $(N \times P)$. The second equation (15) represents the nonlinear and interactive transformation of features as they pass through the hidden layer neurons, where $s^l_{i,t-1}$ $(N \times d_l)$ are temporal values, d_l is the number of hidden units at layer l, and W^{l-1} $(d_l \times d_{l-1})$ is weight matrix. g_3 is an activation function. Last equation (16) describes how factor betas $\beta_{i,t-1}$ $(N \times K)$emerge from the terminal output layer with W^{L_β} $(K \times d_{L_\beta})$ as the final layer weight matrix.

2.3.3. Combined Model

The architecture of our sequence network model, illustrated in Figure 4, merges two modules: the beta network models the nonlinear relationship between factor loadings and asset characteristics; the factor network captures latent factors using periodic return inputs and hidden states. Then we combine the beta functions and latent factor vectors, generating the network's outputs.

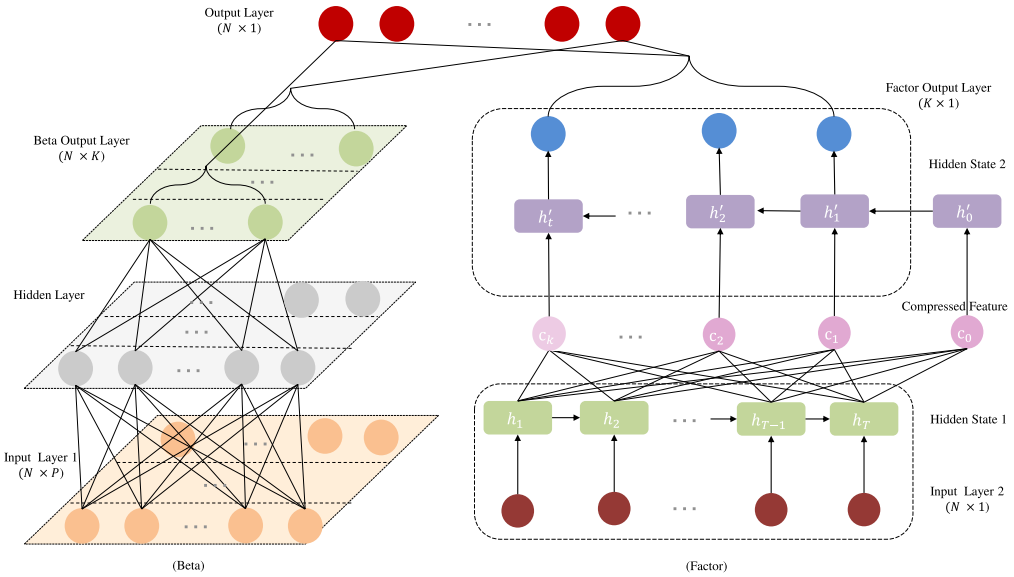

Figure 4: The architecture of Sequence Model with Attention-Mechanism.

We follow Gu et al. (2021) to construct the beta network (left) which uses a feedforward neural network with one hidden layer (grey) to determine how factor loadings $\beta_{i,t-1}$ (green) are generated by firm characteristics $z_{i,t-1}$ (orange). The factor network(right) shows how latent factors f_t (blue) are derived from compressed features c_t (pink). These features are extracted from asset returns r_t (red), along with its hidden state h_t, and the output layer's hidden state h'_t. We also replace the individual returns with a P-dimensional vector of dynamically re-weighted, characteristic-based portfolios, $x_t = (Z'_{t-1}Z_{t-1})^{-1}Z'_{t-1}r_t$, to reduce the model's complexity. This method offers three benefits: a significant reduction in the sequence model's parameters due to smaller dimensions (P versus N), an effective bypass of return panel imbalances from missing stocks, and enhanced latent factor determination, leveraging the proven importance of characteristic-managed portfolios in refining conditional linear factor models.

2.4. Estimation

Similar to Kelly et al. (2019) and Gu et al. (2021), our aim is to optimize the following function:

$$\mathcal{L}\big(r_{i,t};\theta\big) = \frac{1}{N}\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\left\|r_{i,t} - \beta_{i,t-1}f_t\right\|_2^2 \tag{17}$$

where $\mathcal{L}\left(r_{i,t};\theta\right)$ is the objective function with θ including weight and bias parameters in beta and factor networks. In order not to lose generality, we assume the input of the factor network is $r_{i,t}$, since a similar estimation procedure holds when the input is $x_{i,t}$.

To optimize $\mathcal{L}\left(r_{i,t};\theta\right)$, gradient descent (GD), which minimizes a function by iteratively moving in the direction of the steepest descent, is the common method. However, GD requires the entire dataset to compute the gradient at each step, it can be very slow and impractical for large datasets. To solve above problem, we apply stochastic gradient descent (SGD), which improves upon the idea of GD by using only one or a few training examples at a time to calculate the gradient. This means SGD doesn't require the entire dataset to update the model's parameters. We utilize the adaptive moment estimation (Adam) algorithm Kingma and Ba (2014) for gradient evaluation, applying it to minibatches of size M from the full data in each iteration. The Adam algorithm combines ideas from two other extensions of SGD to compute adaptive learning rates for each parameter. It avoids fluctuation when using standard SGD. Through Adam algorithm, we construct our estimators θ as follows.

$$\theta = argmin_{\theta}\left(\sum_{t=1}^{T}\sum_{i=1}^{N}\|r_{i,t} - \beta_{i,t-1}f_t\|_2^2\right) \tag{18}$$

2.4.1. Regularization Implementations

Neural networks often face overfitting and local optima challenges due to their complex structures and numerous parameters. To tackle these problems, it's crucial to use regularization techniques. These measures help to prevent overfitting by adding constraints to the optimization process, thus making the network more robust.

To prevent overfitting, we employ cross-validation, dividing the dataset into training, validation, and testing segments while maintaining temporal sequence. The training set is for model training, the validation set aids

in parameter estimation and hyperparameter tuning, and the testing set assesses the model's performance in out-of-sample scenarios.

The second method to avoid over-fitting is to use regularization method. We append penalty functions to the objective function to avoid overfitting. The penalization effectively diminishes the model's fitting of noise, while maintaining its ability to fit the signal accurately. The most common penalty is to use LASSO or l_1 penalization.

$$\mathcal{L}(r_{i,t};\theta) = \frac{1}{N}\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\left\|r_{i,t} - \beta_{i,t-1}f_t\right\|_2^2 + \lambda\sum_{j}\left|\theta_j\right| \tag{19}$$

where $\lambda\sum_j|\theta_j|$ is l_1 penalization and λ is the regularization parameter. The l_1 penalization, as used in LASSO, zeroes out coefficients of certain covariates, promoting sparse weight parameters and suppressing insignificant ones. A validation sample aids in setting the non-negative hyperparameter, λ.

Our third optimization technique is early stopping. It begins with initial, simple parameterization (θ values near zero) and updates these parameters to reduce training errors. Concurrently, it estimates errors on a validation sample, halting optimization when they rise, typically before minimizing training errors. This approach, which prevents overfitting by stopping parameter adjustment early, is a favored alternative to "l_2" penalization due to its lower computational cost and effective regularization.

3. Empirical Result of US Equity Market

3.1. Data

We apply our model to analyze the US equity market. We collect monthly individual stock returns from CRSP and use monthly Treasury bill rates from the Kenneth French Data Library as the risk-free rate to calculate excess returns. The returns dataset is from January 1980 to December 2023 for all firms listed in the three major exchanges: NYSE, AMEX, and NASDAQ. In addition, we collect 64 different predictive characteristics for each stock. All listed characteristics are referred to Gu et al. (2021); Freyberger et al. (2020); Chen et al. (2023). All data could be constructed from the CRSP/Compustat database. In all characteristics, 38 are updated annually, 7 are updated quarterly, and 19 are updated monthly. We match stock characteristics with corresponding returns to eliminate forward-looking bias in our dataset. Different from Gu et al. (2021), we adopt a moving window

method to capture the time-varying market states. The moving window has a length of 25 years, split by a 20-year training sample, 4-year validation sample, and 1-year test sample, starting from January 1980. Therefore, the moving window rolls 20 times up to December 2023, yielding a total out-of-sample period of 20 years to test our empirical asset pricing models. For the prevalent issue of missing data, we also substitute missing returns or characteristics with their cross-sectional medians.

For each stock i at month t, $r_{i,t}$ represents the return, and $z^*_{ij,t}$, $j = 1, \ldots, 64$, denotes 64 distinct characteristics. To counteract the skewness and leptokurtosis in asset characteristics observed by Gu et al. (2021), and mitigate outlier effects, we employ their recommended rank normalization method:

$$z_{ij,t} = \frac{2}{N+1} rank(z^*_{ij,t}) - 1 \tag{20}$$

where $rank\left(z^*_{ij,t}\right)$ is the rank of $z^*_{ij,t}$ within all characteristics $\left\{z^*_{ij,t}\right\}$. After rank-normalization, $z_{ij,t}$ values fall within $(-1,\ 1)$. We analyze using the vector $z_{i,t}$, composed of normalized asset characteristics, where each j-th entry is $z_{ij,t}$.

3.2. Model Comparison

In our empirical analysis, we compare several classical models, machine learning models and deep learning models. For classical models, we use ordinary least squares (OLS) and IPCA. For machine learning models, we use random forest (RF). For deep learning model, we use traditional Neural Networks (NN), CAE, Generative Adversarial Networks (GAN) and our designed models. We restrict OLS to a sparse parameterization by forcing the model to include only three covariates (size, value, and momentum). The IPCA model, introduced by Kelly et al. (2019), is based on a linear factor structure with conditional betas linearly related to covariates. RF uses an ensemble of decorrelated decision trees to capture nonlinearities and interaction effects. NN model aggregates and nonlinearly transform input signals through multiple layers, enabling rich hierarchical representations and allowing complex interactions among predictors to emerge endogenously. The CAE model, introduced by Gu et al. (2021), uses the Autoencoder and neural network for latent factors and nonlinear factor loading, respectively. We use their CA_2 model, which has two hidden layers in their beta network. The CA_2 model has the best overall performance in both statistical and economic estimation

in Gu et al. (2021). GAN model, which is used by Chen et al. (2023), applies adversarial learning method to train nonlinear neural networks with an objective function of weighted sample moments, which also guarantee the absence of arbitrage. Our models inlude the following three models: SeqRAT, SeqGAT and SeqLAT.They use the neural networks to learn the factor loadings and the encoder-decoder structure to learn the latent factors. In factor network part, SeqRAT uses traditional RNN, SeqGAT replaces RNN with GRU and SeqLAT replaces RNN with LSTM to solve gradient vanishing problems. Besides, we use a one-layer specification on the factor network and the only variation is the number of neurons, which is from 1 to 6. The number of neurons in factor network will correspond to the number of factors in the model.

3.3. Statistical Performance

We follow Kelly et al. (2019) and Gu et al. (2021) to use total and predictive R^2 to evaluate the model's capacity to explain stock risk and predict future returns, with total R^2 assessing contemporaneous factor identification and predictive R^2 gauging accuracy in risk compensation variation, respectively.

$$R^2_{total} = 1 - \frac{\sum \left(r_{i,t} - \widehat{\beta'}_{i,t-1}\widehat{f_t}\right)^2}{\sum r^2_{i,t}} \tag{21}$$

and

$$R^2_{pred} = 1 - \frac{\sum \left(r_{i,t} - \widehat{\beta'}_{i,t-1}\widehat{\mu_{t-1}}\right)^2}{\sum r^2_{i,t}} \tag{22}$$

where μ_{t-1} is the prevailing sample average of f_t up to month $t-1$.

Our out-of-sample total R^2 results in Table 1 demonstrate that incorporating sequential structure and attention leads to substantial gains in explanatory power relative to both classical factor models and machine learning models. At the individual returns level, our Sequence models achieve higher total R^2, followed by GAN with 27.92% and CAE model with 28.56%. Among all benchmarks, we notice that IPCA still delivers a strong total R^2 for both individual stocks r_t and managed portfolios x_t, consistently outperforming OLS, and RF, and even the NN model at some values of K. This indicates that enforcing a linear latent-factor structure with characteristics based betas sometimes remains an effective way to organize cross-sectional variation. Nevertheless, the magnitude of improvement generated by IPCA

is surpassed by our sequence-attention architectures. For example, SeqLAT explains roughly 42% of individual stock variation across all K values, compared with only 28% for CAE and 23% for IPCA at $K = 6$. The gap widens further at the portfolio level: SeqLAT attains 70.54% total R^2 when $K = 6$, significantly exceeding CAE (58.85%) and IPCA (53.28%). These results indicate that the sequence-based models capture systematic structure in returns that neither nonlinear models nor linear latent-factor methods can extract.

Table 1: Total R^2 (%)

Model	Test assets	*Number of Factors(K)*					
		1	2	3	4	5	6
OLS	$\boldsymbol{r}_t$	5.82	4.63	3.42	1.03	-0.06	-2.42
	$\boldsymbol{x}_t$	30.15	35.48	40.85	43.42	45.35	47.53
IPCA	$\boldsymbol{r}_t$	12.02	16.02	18.54	19.76	21.43	23.11
	$\boldsymbol{x}_t$	41.43	45.62	48.07	50.32	52.19	53.28
RF	$\boldsymbol{r}_t$	10.26	12.02	13.71	15.42	17.17	18.26
	$\boldsymbol{x}_t$	35.40	40.32	42.45	45.61	47.73	50.77
NN	$\boldsymbol{r}_t$	15.19	18.45	20.77	21.42	22.05	22.43
	$\boldsymbol{x}_t$	40.02	43.85	46.64	51.32	55.47	57.93
CAE	$\boldsymbol{r}_t$	16.27	21.82	22.76	25.52	27.13	28.56
	$\boldsymbol{x}_t$	43.84	46.17	48.09	51.02	55.36	58.85
GAN	$\boldsymbol{r}_t$	20.18	22.23	23.81	25.45	26.62	27.92
	$\boldsymbol{x}_t$	56.17	58.62	60.54	60.26	62.07	62.83
SeqRAT	$\boldsymbol{r}_t$	35.09	35.92	36.24	36.84	37.25	37.02
	$\boldsymbol{x}_t$	59.56	60.28	62.98	64.32	66.73	67.56
SeqGAT	$\boldsymbol{r}_t$	39.94	40.85	41.94	41.98	42.12	42.17
	$\boldsymbol{x}_t$	61.73	63.79	65.62	68.69	69.67	70.34
SeqLAT	$\boldsymbol{r}_t$	40.98	41.35	41.77	42.03	42.54	42.65
	$\boldsymbol{x}_t$	63.53	65.03	67.51	68.71	70.39	70.54

Within the family of sequence models, the three architectures display markedly different scaling behavior as K increases. SeqLAT delivers the highest total R^2 for $\boldsymbol{r}_t$ in most specifications and shows a clear upward trajectory with larger K, indicating that the LSTM–attention combination is particularly effective at allocating additional latent-factor capacity toward learning persistent, time-varying structure in returns. SeqRAT, by contrast, exhibits a nearly flat profile across K: its explanatory power clusters around

36%, suggesting that the simpler recurrent architecture reaches its effective capacity quickly and extracts limited incremental information from additional factors. SeqGAT lies between these two, improving meaningfully with K but not matching the robustness of SeqLAT. Most importantly, none of the sequence models display the severe deterioration seen in OLS at high K. The smooth, monotonic gains for SeqLAT underscore its ability to convert increased dimensionality into economically meaningful signal extraction rather than overfitting idiosyncratic noise—a property that distinguishes sequence-attention mechanisms from both static and purely recurrent alternatives.

The predictive R^2 results in Table 2 sharpen the distinctions across model classes. Static benchmarks perform weakly out of sample: OLS provides essentially no predictive content, with its R^2 for $\boldsymbol{r}_t$ turning negative at higher K, and RF, NN remain confined about 2% for $\boldsymbol{r}_t$ and below 3% for $\boldsymbol{x}_t$. IPCA delivers meaningful improvements, reaching 3.56% for $\boldsymbol{r}_t$ and 4.06% for $\boldsymbol{x}_t$ at $K = 6$, but these gains are small relative to its strong total fit, indicating that the structure it captures is largely cross-sectional rather than intertemporal.

The sequence models display a markedly different pattern. Predictive performance rises sharply with K, underscoring the importance of exploiting temporal dependence. SeqRAT increases from 2.81% to 5.85% for $\boldsymbol{r}_t$, and from 3.27% to 6.15% for $\boldsymbol{x}_t$. SeqGAT performs still better, reaching 6.09% and 6.51% at $K = 6$. SeqLAT achieves the highest accuracy across specifications, attaining 6.17% for individual returns and 7.15% for portfolios. These values represent improvements of more than 50% over IPCA and far exceed the other benchmark models.The results show that the sequence attention architectures convert additional factor dimensionality into economically meaningful predictive gains. By capturing dynamic and nonlinear interactions that traditional models systematically miss, these models deliver a substantial improvement in the ability to forecast both individual returns and portfolio-level outcomes.

Finally, both tables highlight how model performance evolves with the number of latent factors and the choice of test assets. While most models benefit from larger K, the gains are uneven: OLS deteriorates sharply at higher K due to overfitting, whereas the sequence-attention models maintain or improve predictive performance monotonically, reflecting their capacity to regularize and allocate latent-factor complexity effectively. Moreover, shifting from predicting individual stocks r_t to predicting managed portfolios x_t yields large increases in both total and predictive R^2 across all models, consistent with diversification reducing idiosyncratic noise. Importantly, Se-

Table 2: Predictive R^2 (%)

Model	Test assets	*Number of Factors(K)*					
		1	2	3	4	5	6
OLS	$\boldsymbol{r}_t$	0.56	0.37	0.10	0.04	-0.16	-0.34
	$\boldsymbol{x}_t$	0.83	0.92	1.03	0.81	0.85	0.77
IPCA	$\boldsymbol{r}_t$	2.06	2.52	2.63	2.85	3.22	3.56
	$\boldsymbol{x}_t$	2.62	2.93	3.01	3.18	3.56	4.06
RF	$\boldsymbol{r}_t$	1.35	1.54	1.82	1.76	1.87	1.96
	$\boldsymbol{x}_t$	1.45	1.56	1.72	1.50	1.76	1.87
NN	$\boldsymbol{r}_t$	1.84	1.95	2.11	2.09	2.14	2.21
	$\boldsymbol{x}_t$	2.39	2.54	2.71	2.68	2.74	2.83
CAE	$\boldsymbol{r}_t$	2.13	2.45	2.62	2.71	2.74	2.85
	$\boldsymbol{x}_t$	2.53	2.89	3.12	3.22	3.27	3.39
GAN	$\boldsymbol{r}_t$	1.93	1.98	2.09	2.03	2.11	2.14
	$\boldsymbol{x}_t$	2.47	2.73	2.85	2.79	2.88	3.02
SeqRAT	$\boldsymbol{r}_t$	2.81	3.62	4.43	5.27	5.18	5.85
	$\boldsymbol{x}_t$	3.27	3.89	4.92	5.35	5.81	6.15
SeqGAT	$\boldsymbol{r}_t$	3.12	4.38	5.12	5.64	5.77	6.09
	$\boldsymbol{x}_t$	3.65	4.72	5.97	6.04	6.11	6.51
SeqLAT	$\boldsymbol{r}_t$	3.31	4.98	5.46	6.01	5.87	6.17
	$\boldsymbol{x}_t$	3.87	4.62	5.52	6.75	7.07	7.15

qLAT and SeqGAT exhibit the largest incremental gains at the portfolio level, suggesting that their dynamically estimated factors align closely with the common return components that matter for portfolio level predictability. This property is essential for asset pricing applications, as it implies that the models are not merely fitting stock-specific noise but are capturing systematic drivers of returns that translate into superior performance in tradable portfolios.

3.4. Economic Performance

We also evaluate all models economically by examining Sharpe ratios of portfolios, informed by their predictions on the conditional mean and variance of returns in the testing sample. We sort stocks into deciles, forming two portfolio types: a long-short portfolio from buying top 10% (decile 10) and selling bottom 10% (decile 1), and a long-only portfolio comprising only the top 10% (decile 10), using equal weighting and value weighting for both.

For equal-weighted portfolios in Table 3, models with sequential structure and attention clearly dominate the benchmarks. OLS delivers negative Sharpe ratios for both long-short and long-only portfolios across all K, underscoring that naïvely projecting returns on characteristics without a disciplined factor structure destroys rather than creates investable signal. IPCA, RF, NN, and CAE perform progressively better as K rises, but their long-short Sharpe ratios remain well below those of the sequence models. In contrast, SeqLAT and SeqGAT generate Sharpe ratios that are an order of magnitude larger than IPCA for small K, and continue to improve as K increases. SeqLAT typically achieves the highest Sharpe ratios across K for both long-short and long-only portfolios, with SeqGAT essentially tied in several specifications, especially at low and intermediate K. This pattern suggests that the LSTM-plus-attention architecture is particularly effective at converting cross-sectional predictability into realized risk-adjusted returns, while the gated-attention variant provides a competitive alternative when factor capacity is more tightly constrained.

The value-weighted results in Table 4 provide a more stringent test of economic relevance, as they place greater weight on large, more liquid stocks and thereby reduce the influence of small-cap names where statistical predictability is often strongest but harder to monetize at scale. Under this more conservative weighting scheme, the broad ranking of models is preserved. OLS continues to yield negative Sharpe ratios in most configurations, and IPCA

Table 3: Equal Weights: Out-of-sample Sharpe ratios of long–short and long–only portfolios without transaction costs

Long–short portfolios

Model	1	2	3	4	5	6
OLS	-0.49	-0.71	-0.32	-0.24	0.09	-0.07
IPCA	0.20	0.46	0.85	1.25	1.76	1.65
RF	0.38	0.48	0.63	0.79	1.03	1.01
NN	0.52	0.78	1.37	1.71	1.78	1.86
CAE	0.26	0.81	1.42	1.53	2.47	2.40
GAN	0.67	0.92	1.74	2.10	2.34	2.64
SeqRAT	0.71	0.98	1.52	1.93	2.39	2.56
SeqGAT	0.91	0.89	1.64	2.15	2.73	2.87
SeqLAT	0.94	1.36	1.82	2.23	2.85	3.03

Long–only portfolios

Model	1	2	3	4	5	6
OLS	-0.64	-0.86	-0.41	-0.33	-0.05	-0.11
IPCA	0.23	0.52	0.77	1.19	1.42	1.32
RF	0.27	0.35	0.57	0.83	1.02	0.95
NN	0.47	0.68	0.86	1.31	1.72	1.87
CAE	0.53	0.85	1.21	1.67	2.12	1.92
GAN	0.59	0.87	1.41	1.65	1.83	2.07
SeqRAT	0.68	0.93	1.48	1.71	1.95	2.09
SeqGAT	0.78	0.96	1.53	1.75	2.16	2.24
SeqLAT	0.82	1.19	1.64	1.83	2.20	2.39

only gradually transitions from negative to modestly positive Sharpe as K increases. Other deep learning models (NN, CAE, GAN) improve on IPCA and deliver reasonable positive Sharpe ratios for long-short portfolios at higher K, but their long-only performance remains only moderately above zero, indicating limited scope for implementing these signals in long-only mandates. By contrast, the sequence-attention models maintain a clear edge: SeqLAT again delivers the highest or near-highest Sharpe ratios across K and portfolio types, with SeqGAT very close and occasionally slightly ahead at some intermediate factor counts. Importantly, the sequence models remain robustly

profitable even for value-weighted, long-only portfolios, where Sharpe ratios reach levels that are economically meaningful after accounting for realistic transaction costs and capacity constraints.

Table 4: Value Weights: Out-of-sample Sharpe ratios of long–short and long–only portfolios without transaction costs

Long–short portfolios

Model	1	2	3	4	5	6
OLS	-0.94	-1.35	-0.77	-0.45	-0.23	-0.40
IPCA	-0.13	-0.05	0.32	0.58	0.93	1.02
RF	-0.17	-0.03	0.26	0.52	0.77	0.91
NN	-0.11	0.14	0.53	0.74	1.14	1.37
CAE	-0.05	0.21	0.67	0.92	1.46	1.45
GAN	-0.02	0.26	0.70	1.03	1.41	1.57
SeqRAT	0.02	0.31	0.76	1.10	1.46	1.60
SeqGAT	0.02	0.35	0.83	1.17	1.53	1.79
SeqLAT	0.03	0.34	0.82	1.23	1.56	1.83

Long–only portfolios

Model	1	2	3	4	5	6
OLS	-1.19	-1.36	-1.26	-0.97	-0.81	-1.12
IPCA	-0.46	-0.22	0.09	0.27	0.51	0.73
RF	-0.37	-0.11	0.15	0.47	0.32	0.66
NN	-0.18	0.32	0.44	0.64	0.81	0.93
CAE	0.24	0.43	0.61	0.86	1.05	0.96
GAN	0.29	0.44	0.72	0.85	0.93	1.04
SeqRAT	0.33	0.48	0.74	0.88	0.97	1.05
SeqGAT	0.41	0.52	0.83	0.94	1.15	1.13
SeqLAT	0.43	0.57	0.85	0.96	1.17	1.24

Both the equal-weighted and value-weighted evidence demonstrates that the advantages of our sequence-attention architectures are not confined to statistical measures or to small, illiquid stocks. Moreover, the near-monotonic improvement in Sharpe ratios with K for SeqLAT and SeqGAT stands in stark contrast to the instability of OLS and the plateauing of IPCA and CAE, indicating that the sequential architectures can effectively exploit addi-

tional factor capacity. Overall, these results suggest that modeling the joint time-series and cross-sectional dynamics of characteristics with attention-based sequence networks yields signals that are both statistically robust and economically exploitable in realistic portfolio management settings.

3.5. Characteristics importance

To understand the drivers of predictive performance across all models, we examine the relative importance of the asset characteristics. The results are shown in Figure 5. Our analysis yields two main findings. First, a small set of characteristics , primarily short and intermediate horizon momentum such as 1-month momentum(mom1m), Change in mom 6(chmom), return and idiosyncratic volatility such as Return volatility(retvol) and Idiosyncratic return volatility(idiovol), liquidity related volatility measures like Volatility of liquidity (share turnover)(stdturn), and size (mvel1). These characteristics consistently dominate the explanatory power across all specifications. Second, and more importantly, the ranking of characteristic importance changes systematically when moving to our sequential models with time seires information and temporal dependence. These patterns highlight the economic relevance of dynamic information in the return generating process.

Traditional classical models and machine learning models, including IPCA, RF, NN, and CAE, tend to assign relatively higher importance to characteristics whose predictive content derives from cross sectional levels rather than temporal evolution. For example, mvel1, maximum daily return (maxret), and dollar volume (dolvol) consistently rank highly in these models. This reflects the fact that those architectures extract predictive content from contemporaneous signals, and therefore favor characteristics that proxy for structural cross-sectional effects such as firm scale, liquidity segmentation, or jump risk.

Momentum characteristics remain important in those models, but primarily through their current levels rather than the shape of the momentum trajectory. Similarly, volatility-related characteristics such as retvol and idiovol contribute predictive power but cannot be exploited for their temporal persistence or interaction with market regimes. As a result, those models attribute more weight to characteristics with strong unconditional cross sectional explanatory power, consistent with foundings from.

When we move eyes to sequential models, the characteristic importance patterns shift markedly. These models consistently increase the importance assigned to characteristics whose predictive content depends on temporal

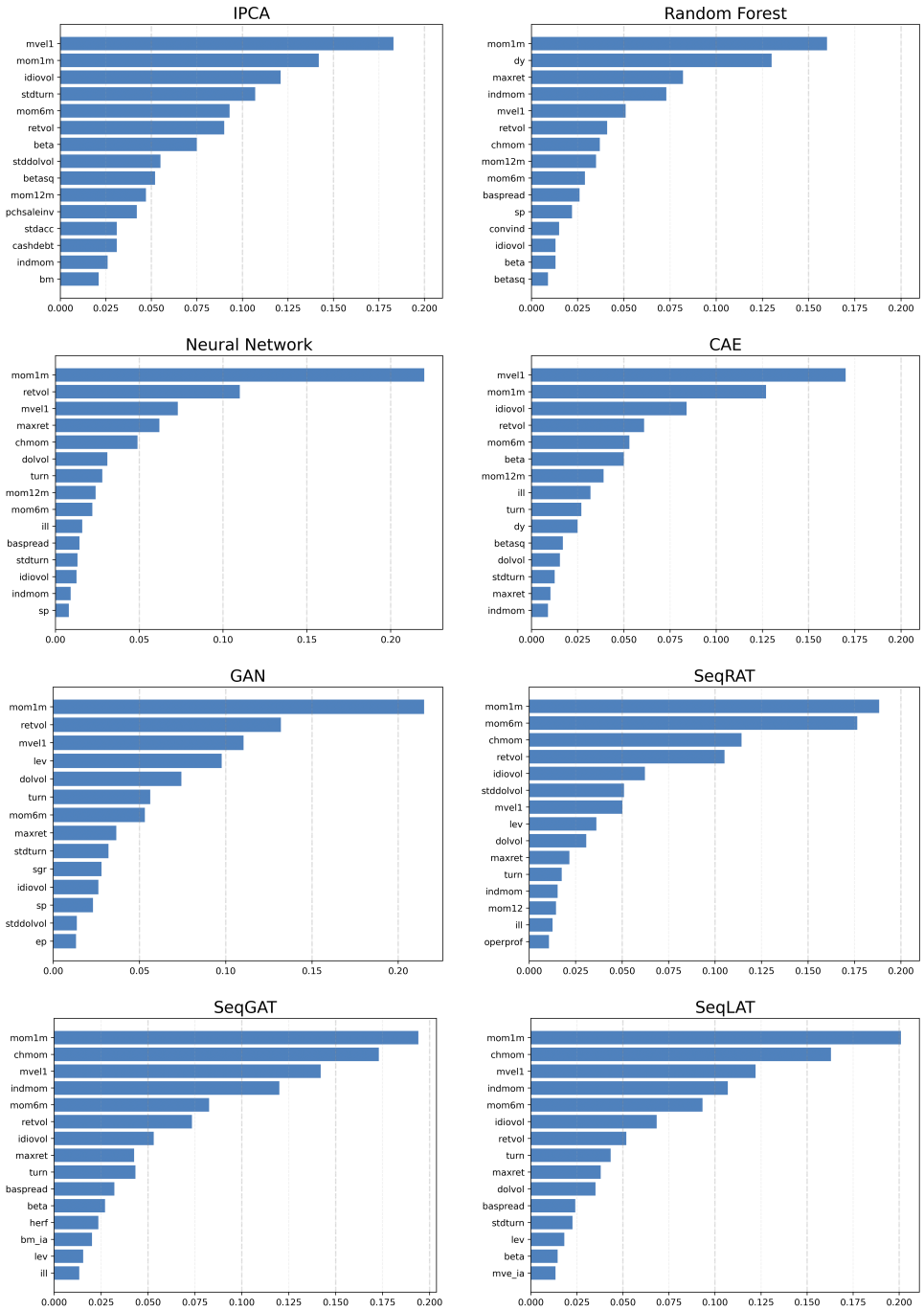

Figure 5: Top 15 Characteristics Importance for all models

dynamics, including momentum evolution, volatility clustering, and liquidity deterioration.

First, we notice short and intermediate horizon momentum (mom1m, chmom, mom6m) become substantially more important in SeqRAT, SeqGAT, and SeqLAT. Sequential models are able to condition on the entire return trajectory, allowing them to exploit information embedded in momentum curvature, recent accelerations, and regime transitions. The stronger emphasis on chmom in sequential models reflects their ability to detect changes in the slope of the trend. Except sequential models, only NN and GAN exhibit some sensitivity to chmom.

Volatility based characteristics (retvol, idiovol) and liquidity-volatility measures (stdturn, stddolvol) also become significantly more important in sequential settings. Volatility and liquidity exhibit clustering and regime shifts that static models cannot exploit. Sequential models condition on the trajectory and timing of volatility shocks, which becomes especially valuable in stressed markets. Attention layers further concentrate model capacity on episodes where volatility shocks convey more information about expected returns, increasing the contribution of these characteristics to overall predictive accuracy.

Besides, the relative importance of characteristics varies across SeqRAT, SeqGAT, and SeqLAT in a manner consistent with their memory structures. RNNs tend to prioritize recent shocks, GRUs adaptively smooth noisy dynamics, and LSTMs retain information over longer horizons. Consequently, SeqLAT places the highest weight on intermediate horizon momentum and slowly evolving volatility patterns, while SeqRAT emphasizes near term signals such as short horizon momentum and recent volatility spikes. These differences highlight the implication of distinguishing between short and long memory components of characteristic dynamics.

Overall, these findings imply that the better performance of sequential models does not arise simply from greater architectural complexity, but rather from their ability to reinterpret traditional characteristics as dynamic signals. Characteristics such as momentum, volatility, and turnover acquire predictive power primarily through their time path, not just their level. traditional linear models and machine learning models, by construction, cannot access these channels. The evidence demonstrates that capturing temporal dependence alters the structure of characteristic importance.

3.6. Model Significance and Correlation

Moreover, we use the Jobson and Korkie (1986) significance test with Memmel (2003) correction to compare all models. The test is also used in Ao et al. (2018); Caner et al. (2023). The null and alternative hypothesis are as follows:

$$H_0 : SR_{seqlat} \leq SR_0$$
$$H_1 : SR_{seqlat} > SR_0$$

where SR_{seqlat} is the Sharpe ratio of SeqLAT and SR_0 represents Sharpe ratio from remaining models. We set SeqLAT with $K = 6$ as the benchmark to test against all other methods with $K = 6$ since it generally shows the best Sharpe ratios in both long-short and long-only portfolios.

Table 5: P-values of the Jobson–Korkie Test with Memmel Correction

Compared Model	**Long–Short**	**Long–Only**
OLS	0.004	0.008
IPCA	0.028	0.031
RF	0.014	0.010
NN	0.037	0.056
CAE	0.131	0.087
GAN	0.108	0.128
SeqRAT	0.443	0.325
SeqGAT	0.518	0.463

Table 5 reports the Jobson–Korkie test with the Memmel correction comparing the Sharpe ratio of SeqLAT against all other methods. The results show a clear separation between SeqLAT and traditional models. SeqLAT significantly outperforms OLS, IPCA, RF, and NN in both long–short and long–only settings, with p-values ranging from 0.004 to 0.056. These p-values reject the null hypothesis at conventional significance levels, indicating that SeqLAT delivers reliably higher risk-adjusted performance relative to all non-sequential benchmarks.

The CAE and GAN models exhibit weaker evidence against the null, with p-values between 0.087 and 0.131 in both conditions. Although these values do not cross the 5% threshold, they still suggest that SeqLAT tends to

outperform nonlinear architectures that rely only on cross-sectional transformations. Importantly, SeqRAT and SeqGAT yield relatively large p-values which os from 0.325 to 0.518, reflecting that their Sharpe ratios are statistically indistinguishable from that of SeqLAT. This pattern is consistent with the idea that the three sequence models capture a similar form of temporal dependency, whereas static and feed-forward models fail to exploit such information. The significance tests confirm that SeqLAT's performance advantage arises primarily in contrast to traditional and cross sectional learning methods, while remaining comparable to other temporal architectures.

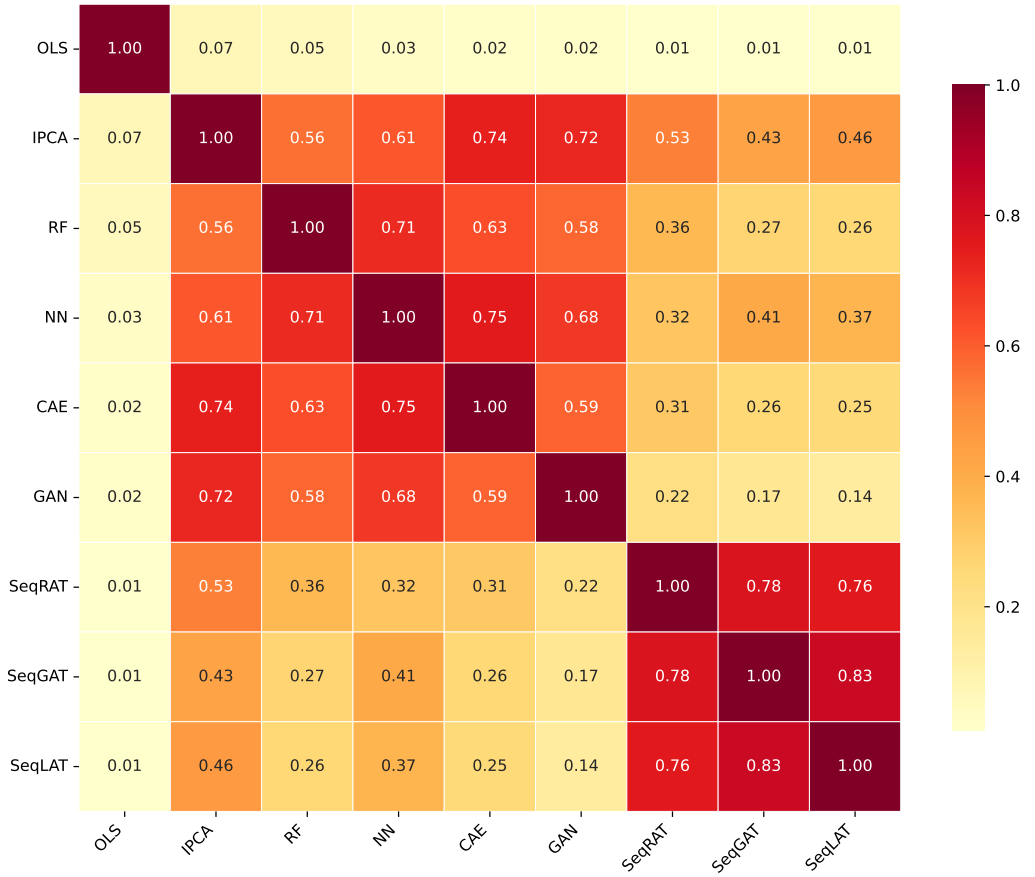

Figure 6: Correlation Matrix of Strategy Returns

Figure 6 further reinforces the conclusions from the Sharpe ratio tests. We compute the Spearman correlation of strategy returns across all avail-

able out of sample dates, ensuring that each model is evaluated on the same set of periods. Traditional models (IPCA, RF, NN, CAE, GAN) exhibit strong comovement, with correlations typically between 0.56 and 0.75. This high internal clustering indicates that, despite architectural differences, these methods extract largely overlapping cross sectional return premia, consistent with their similar Sharpe ratios and the significant p-values obtained in comparisons with SeqLAT.

In contrast, SeqLAT, SeqRAT and SeqGAT display a distinctly different correlation profile. Their correlations with traditional approaches are extremely low, while showing strong mutual dependence among themselves. This pattern suggests that the sequence architectures rely on a temporal predictive structure fundamentally different from the static or feed forward transformations used by classical machine learning methods. In line with the significance test results, the low cross model correlation indicates that our models capture complementary and largely orthogonal signals rather than repackaging information already embedded in conventional predictors.

3.7. Transcation Costs Analysis

In this section, we incorporate a transaction cost adjustment of 30 basis points rebalancing. The Sharpe ratios reported in Tables 6 and 7 reveal several economically meaningful patterns once these trading frictions are taken into account.

After costs, Sharpe ratios decline across all models and portfolio constructions, though the magnitude of the reduction varies systematically with model structure, portfolio weighting, and factor dimensionality. Equal weighted portfolios, which inherently require more substantial rebalancing, exhibit more pronounced decreases, particularly for models whose forecasts adapt more aggressively to the time series evolution of firm characteristics. In this respect, the SeqLAT and SeqGAT architectures naturally incur higher turnover due to their ability to extract and exploit rapidly evolving information from characteristic histories. Nevertheless, even after conservatively applying transaction costs, these sequence-based models continue to deliver Sharpe ratios well above those of all benchmark models. For equal-weighted long-short portfolios, SeqLAT consistently preserves Sharpe ratios in the range of 1.4 to 1.9 for $K \geq 4$, indicating that their forecasting gains comfortably outweigh the associated trading intensity. By contrast, traditional models such as OLS and IPCA generate negative or near-zero Sharpe ratios

once costs are applied, highlighting that much of their apparent statistical performance is not readily monetizable.

A more stringent assessment of implementability arises under value weighted portfolio construction, which better reflects the trading environment faced by large institutional allocators. In this setting, the effect of transaction costs is considerably muted: the emphasis on large cap stocks reduces proportional rebalancing needs and significantly lowers turnover. Under these conditions, the cost-adjusted performance of the sequence models remains robust. SeqLAT and SeqGAT continue to dominate the alternative approaches, maintaining friction adjusted Sharpe ratios from 1.0 to 1.6 for long-short portfolios and delivering materially positive long-only performance. Importantly, the relative ranking of models is largely preserved before and after trading cost adjustments, suggesting that the improvements offered by the sequence architectures stem from genuine enhancements in signal quality rather than from exposure to illiquid segments of the cross-section or mechanically high turnover.

This section demonstrates that the economic value of the sequence based models survives conservative frictional adjustments and remains substantially above that of conventional linear or nonlinear benchmarks. Although these architectures naturally induce greater trading activity due to their responsiveness to intertemporal variation in characteristics, the incremental predictive accuracy they deliver more than compensates for the associated costs. The persistence of outperformance under both equal and value weighted constructions indicates that the signals extracted by SeqLAT and SeqGAT translate into implementable and scalable portfolio improvements, reaffirming the economic relevance of modeling characteristic dynamics through attention based sequence learning.

4. Conclusion

Our paper introduces a nonlinear conditional asset pricing framework that embeds a sequence model with attention into a latent factor structure. The factor network employs recurrent architectures to capture the temporal evolution of latent factors, while the beta network maps firm characteristics into time varying exposures through a flexible nonlinear transformation. By combining recurrent "memory" with an attention mechanism, the model exploits long and short horizon dependence in return dynamics, addressing

Table 6: Equal Weights: Out-of-sample Sharpe ratios with 30bp monthly transaction cost

Long–short portfolios

Model	1	2	3	4	5	6
OLS	-0.58	-0.82	-0.37	-0.29	-0.02	-0.14
IPCA	0.18	0.40	0.75	1.10	1.54	1.45
RF	0.30	0.38	0.50	0.63	0.82	0.81
NN	0.41	0.60	1.07	1.33	1.39	1.45
CAE	0.20	0.62	1.08	1.16	1.87	1.82
GAN	0.50	0.69	1.31	1.58	1.76	1.89
SeqRAT	0.50	0.69	1.06	1.35	1.65	1.79
SeqGAT	0.59	0.72	1.10	1.44	1.83	1.91
SeqLAT	0.61	0.88	1.18	1.45	1.85	1.94

Long–only portfolios

Model	1	2	3	4	5	6
OLS	-0.77	-0.93	-0.56	-0.42	-0.13	-0.19
IPCA	0.20	0.46	0.68	1.05	1.25	1.16
RF	0.22	0.28	0.46	0.66	1.01	0.76
NN	0.37	0.53	0.67	1.02	1.34	1.46
CAE	0.40	0.64	0.92	1.27	1.61	1.46
GAN	0.44	0.65	1.06	1.24	1.37	1.51
SeqRAT	0.48	0.65	1.03	1.20	1.37	1.44
SeqGAT	0.52	0.68	1.03	1.17	1.45	1.50
SeqLAT	0.53	0.77	1.06	1.19	1.43	1.55

limitations of traditional linear and machine learning designs that treat each period in isolation.

Methodologically, the attention module shortens the effective gradient path and concentrates on informative segments of the return history, mitigating vanishing gradient issues that impede deep recurrent models on long panels. The architecture therefore extracts persistent state variables and slow-moving components of risk that are difficult for traditional econometric and machine learning benchmarks to recover.

Empirically, the sequence attention models deliver sizable and robust improvements in explanatory and predictive performance. When test assets are

Table 7: Value Weights: Out-of-sample Sharpe ratios with 30bp monthly transaction cost

Long–short portfolios

Model	1	2	3	4	5	6
OLS	-0.99	-1.40	-0.82	-0.50	-0.28	-0.45
IPCA	-0.20	-0.12	0.25	0.51	0.86	0.95
RF	-0.29	-0.15	0.14	0.40	0.65	0.79
NN	-0.25	0.00	0.39	0.60	1.00	1.23
CAE	-0.21	0.05	0.51	0.76	1.30	1.29
GAN	-0.20	0.08	0.52	0.85	1.23	1.39
SeqRAT	-0.20	0.09	0.54	0.88	1.24	1.38
SeqGAT	-0.22	0.11	0.59	0.93	1.29	1.55
SeqLAT	-0.23	0.08	0.56	0.97	1.30	1.57

Long–only portfolios

Model	1	2	3	4	5	6
OLS	-1.24	-1.41	-1.31	-1.02	-0.86	-1.17
IPCA	-0.53	-0.29	0.02	0.20	0.44	0.66
RF	-0.49	-0.23	0.03	0.35	0.20	0.54
NN	-0.32	0.18	0.30	0.50	0.67	0.79
CAE	0.08	0.27	0.45	0.70	0.89	0.80
GAN	0.11	0.26	0.54	0.67	0.75	0.86
SeqRAT	0.11	0.26	0.52	0.66	0.75	0.83
SeqGAT	0.17	0.28	0.59	0.70	0.91	0.89
SeqLAT	0.17	0.31	0.59	0.70	0.91	0.98

individual stock returns, SeqLAT substantially exceeds the performance of benchmark models. Within the sequence family, SeqLAT also scales most effectively with the number of latent factors, showing smooth gains in R^2 while SeqRAT saturates and SeqGAT displays intermediate improvements. These patterns indicate that the LSTM attention combination transforms additional factor capacity into meaningful state-variable information rather than noise fitting. The economic relevance of these statistical gains is reflected in portfolio performance. Across long–short and long-only portfolios, and under realistic transaction cost assumptions, SeqLAT delivers the highest out-of-sample Sharpe ratios among all models considered. Its performance

underscores the value of modeling temporal dependence explicitly in conditional factor structures.

Overall, our findings demonstrate that incorporating sequence dynamics and attention into latent factor pricing models yields a powerful and economically grounded framework for forecasting returns and organizing cross-sectional variation. The architecture respects no-arbitrage structure, scales to long horizon data, and can naturally incorporate richer information sets such as macroeconomic variables or textual features. We view SeqLAT as a prototype for a broader class of sequence-based pricing models, and our results suggest that exploiting "memory" in latent factors represents a promising direction for future research in both academic asset pricing and portfolio management.

References

Ang, A., Liu, J., Schwarz, K., 2020. Using stocks or portfolios in tests of factor models. Journal of Financial and Quantitative Analysis 55, 709–750. doi:10.1017/S0022109019000255.

Ao, M., Yingying, L., Zheng, X., 2018. Approaching Mean-Variance Efficiency for Large Portfolios. The Review of Financial Studies 32, 2890–2919. doi:10.1093/rfs/hhy105.

Bahdanau, D., Cho, K., Bengio, Y., 2014. Neural machine translation by jointly learning to align and translate Available at `https://arxiv.org/abs/1409.0473`.

Bali, T.G., Goyal, A., Huang, D., Jiang, F., Wen, Q., 2020. Predicting corporate bond returns: Merton meets machine learning. Georgetown McDonough School of Business Research Paper , 20–110.

Bansal, R., Yaron, A., 2004. Risks for the long run: A potential resolution of asset pricing puzzles. Journal of Finance 59, 1481–1509.

Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks 5, 157–166.

Bianchi, D., Büchner, M., Tamoni, A., 2021. Bond risk premiums with machine learning. The Review of Financial Studies 34, 1046–1089.

Bicksler, J.L., Jensen, M.C., 1973. Studies in the theory of capital markets. Journal of the American Statistical Association 68, 750.

Bryzgalova, S., Pelger, M., Zhu, J., 2020. Forest through the trees: Building cross-sections of stock returns. Social Science Research Network Available at https://ssrn.com/abstract=3493458.

Büchner, M., Kelly, B., 2022. A factor model for option returns. Journal of Financial Economics 143, 1140–1161. doi:https://doi.org/10.1016/j.jfineco.2021.12.007.

Callot, L., Caner, M., Özlem Önder, A., Ulasan, E., 2021. A nodewise regression approach to estimating large portfolios. Journal of Business and Economic Statistics 39, 520–531. doi:10.1080/07350015.2019.1683018.

Campbell, J., Cochrane, J., 1999. By force of habit: A consumption-based explanation of aggregate stock market behavior. Journal of Political Economy 107, 205–251. doi:10.1086/250059.

Caner, M., Daniele, M., 2023. Deep learning based residuals in non-linear factor models: Precision matrix estimation of returns with low signal-to-noise ratio. arXiv:2209.04512. available at https://arxiv.org/abs/2209.04512.

Caner, M., Medeiros, M., Vasconcelos, G.F., 2023. Sharpe ratio analysis in high dimensions: Residual-based nodewise regression in factor models. Journal of Econometrics 235, 393–417. doi:https://doi.org/10.1016/j.jeconom.2022.03.009.

Chen, L., Pelger, M., Zhu, J., 2023. Deep learning in asset pricing. Management Science doi:10.1287/mnsc.2023.4695. available at https://doi.org/10.1287/mnsc.2023.4695.

Chinco, A., Clark-Joseph, A.D., Ye, M., 2019. Sparse signals in the cross-section of returns. The Journal of Finance 74, 449–492. doi:https://doi.org/10.1111/jofi.12733.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. Available at https://arxiv.org/abs/1406.1078.

Connor, G., Korajczyk, R.A., 1986. Performance measurement with the arbitrage pricing theory: A new framework for analysis. Journal of Financial Economics 15, 373–394. doi:https://doi.org/10.1016/0304-405X(86)90027-9.

Fan, J., Furger, A., Xiu, D., 2015. Incorporating global industrial classification standard into portfolio allocation: A simple factor-based large covariance matrix estimator with high-frequency data. Journal of Business & Economic Statistics 34, 489–503.

Fan, J., Liao, Y., Mincheva, M., 2013. Large covariance estimation by thresholding principal orthogonal complements. Journal of the Royal Statistical Society 75, 603–680.

Fan, J., Mincheva, L.M., 2011. High-dimensional covariance matrix estimation in approximate factor models. Annals of Statistics 39, 3320–3356.

Feng, G., He, J., 2022. Factor investing: A bayesian hierarchical approach. Journal of Econometrics 230, 183–200. doi:https://doi.org/10.1016/j.jeconom.2021.11.001.

Freyberger, J., Neuhierl, A., Weber, M., 2020. Dissecting Characteristics Nonparametrically. The Review of Financial Studies 33, 2326–2377. doi:10.1093/rfs/hhz123.

Gagliardini, P., Ossola, E., Scaillet, O., 2016. Time-varying risk premium in large cross-sectional equity data sets. Econometrica 84, 985–1046. doi:https://doi.org/10.3982/ECTA11069.

Ghysels, E., 1998. On stable factor structures in the pricing of risk: Do time-varying betas help or hurt? The Journal of Finance 53, 549–573. doi:https://doi.org/10.1111/0022-1082.224803.

Gu, S., Kelly, B., Xiu, D., 2020. Empirical Asset Pricing via Machine Learning. The Review of Financial Studies 33, 2223–2273. doi:10.1093/rfs/hhaa009.

Gu, S., Kelly, B., Xiu, D., 2021. Autoencoder asset pricing models. Journal of Econometrics 222, 429–450. doi:https://doi.org/10.1016/j.jeconom.2020.07.009. annals Issue: Financial Econometrics in the Age of the Digital Economy.

He, Zhiguo, Krishnamurthy, Arvind, 2013. Intermediary asset pricing. American Economic Review 103, 732–770.

He, Z., Kelly, B., Manela, A., 2017. Intermediary asset pricing: New evidence from many asset classes. Journal of Financial Economics 126, 1–35. doi:https://doi.org/10.1016/j.jfineco.2017.08.002.

Heaton, J.B., Polson, N.G., Witte, J.H., 2017. Deep learning for finance: deep portfolios. Applied Stochastic Models in Business and Industry 33, 3–12.

Hewamalage, H., Bergmeir, C., Bandara, K., 2021. Recurrent neural networks for time series forecasting: Current status and future directions. International Journal of Forecasting 37, 388–427. doi:https://doi.org/10.1016/j.ijforecast.2020.06.008.

Hochreiter, S., 1991. Untersuchungen zu dynamischen neuronalen netzen. Diploma, Technische Universität München 91, 31.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Computation 9, 1735–1780. doi:10.1162/neco.1997.9.8.1735.

Jiang, J., Kelly, B., Xiu, D., 2023. (re-) imag (in) ing price trends. The Journal of Finance 78, 3193–3249.

Jobson, J.D., Korkie, B., 1986. Performance hypothesis testing with the sharpe and treynor measures: A comment. Journal of Finance 41, 1175–1176.

Ke, Z.T., Kelly, B.T., Xiu, D., 2019. Predicting returns with text data. Technical Report. National Bureau of Economic Research.

Kelly, B., Palhares, D., Pruitt, S., 2023. Modeling corporate bond returns. The Journal of Finance 78, 1967–2008. doi:https://doi.org/10.1111/jofi.13233.

Kelly, B.T., Pruitt, S., Su, Y., 2019. Characteristics are covariances: A unified model of risk and return. Journal of Financial Economics 134, 501–524. doi:https://doi.org/10.1016/j.jfineco.2019.05.001.

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. Computer Science Available at `https://arxiv.org/abs/1412.6980`.

Kitaev, N., Kaiser, Ł., Levskaya, A., 2020. Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451 .

Kozak, S., Nagel, S., Santosh, S., 2020. Shrinking the cross-section. Journal of Financial Economics 135, 271–292. doi:https://doi.org/10.1016/j.jfineco.2019.06.008.

Memmel, C., 2003. Performance hypothesis testing with the sharpe ratio. Finance Letters 1. Available at `Available at SSRN 412588`.

Menzly, L., Santos, T., Veronesi, P., 2004. Understanding predictability. Journal of Political Economy 112, 1–47. doi:10.1086/379934.

Moghar, A., Hamiche, M., 2020. Stock market prediction using lstm recurrent neural network. Procedia Computer Science 170, 1168–1173. doi:https://doi.org/10.1016/j.procs.2020.03.049.

Nagel, S., 2021. Machine learning in asset pricing. volume 8. Princeton University Press.

Pascanu, R., Mikolov, T., Bengio, Y., 2013. On the difficulty of training recurrent neural networks, in: Dasgupta, S., McAllester, D. (Eds.), Proceedings of the 30th International Conference on Machine Learning, PMLR, Atlanta, Georgia, USA. pp. 1310–1318. URL: `https://proceedings.mlr.press/v28/pascanu13.html`.

Ross, S.A., 1976. The arbitrage theory of capital asset pricing. Journal of Economic Theory 13, 341–360.

Sako, K., Mpinda, B.N., Rodrigues, P.C., 2022. Neural networks for financial time series forecasting. Entropy 24. doi:10.3390/e24050657.

Subakan, C., Ravanelli, M., Cornell, S., Bronzi, M., Zhong, J., 2021. Attention is all you need in speech separation, in: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE. pp. 21–25.

Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to sequence learning with neural networks, in: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems 30.

Xuanling Yang, Zhoufan Zhu, D.L., Zhu, K., 2023. Asset pricing via the conditional quantile variational autoencoder. Journal of Business & Economic Statistics 0, 1–14. doi:10.1080/07350015.2023.2223683.

Zhang, H., Goodfellow, I., Metaxas, D., Odena, A., 2019. Self-attention generative adversarial networks, in: International conference on machine learning, PMLR. pp. 7354–7363.

Appendix A: List of stock characteristics

The table lists the acronym and definition for the 42 stock characteristics used as model inputs.

Number	Acronym	Definition	Frequency
1.	absacc	Absolute accruals	Annual
2.	acc	Working capital accruals	Annual
3.	agr	Asset growth	Annual
4.	baspread	Bidask sprea	Monthly
5.	beta	Beta	Monthly
6.	betasq	Beta squared	Monthly
7.	bm	Book to market	Annual
8.	bm_ia	Industry-adjusted book to market	Annual
9.	cash	Cash holdings	Quarterly
10.	cashdebt	Cash flow to debt	Annual
11.	cashpr	Cash productivity	Annual
12.	cfp	Cash flow to price ratio	Annual
13.	cfp_ia	Industry-adjusted cash flow to price ratio	Annual
14.	chatoia	Industry-adjusted change in asset turnove	Annual
15.	chcsho	Change in shares outstanding	Annual
16.	chmom	Change in mom_6	Monthly
17.	chinv	Change in inventory	Annual
18.	chpmia	Industry-adjusted change in profit margin	Annual
19.	chtx	Change in tax expense	Quarterly
20.	convind	Convertible debt indicator	Annual
21.	currat	Current ratio	Annual
22.	depr	Depreciation / PP&E	Monthly
23.	dolvol	Dollar trading volume	Monthly
24.	dy	Dividend to price	Annual
25.	egr	Growth in common shareholder equity	Annual
26.	ep	Earnings to price	Monthly
27.	gma	Gross profitability	Annual
28.	grcapx	Growth in capital expenditures	Annual
29.	herf	Industry sales concentration	Monthly
30.	idiovol	Idiosyncratic return volatility	Monthly
31.	ill	Illiquidity	Monthly

Number	Acronym	Definition	Frequency
32.	indmom	Industry momentum	Monthly
33.	lev	Leverage	Annual
34.	lgr	Growth in long term debt	Annual
35.	maxret	Maximum daily return	Monthly
36.	mom_1	1-month momentum	Monthly
37.	mom_6	6-month momentum	Monthly
38.	mom_12	12-month momentum	Monthly
39.	mvel1	Size	Monthly
40.	mve_ia	Industry-adjusted size	Annual
41.	operprof	Operating profitability	Annual
42.	pchcurrat	Percent change in current ratio	Annual
43.	pchdepr	Percent change in depreciation	Annual
44.	pchquick	Percent change in quick ratio	Annual
45.	pchsaleinv	Percent change in sales-to-inventory	Annual
46.	ps	Financial statements scor	Annual
47.	quick	Quick ratio	Annual
48.	pctacc	Percent accruals	Annual
49.	retvol	Return volatility	Monthly
50.	roa	Return on assets	Quarterly
51.	roavol	Earnings volatility	Quarterly
52.	roe	Return on equit	Quarterly
53.	roic	Return on invested capital	Annual
54.	salecash	Sales to cash	Annual
55.	saleinv	Sales to inventory	Annual
56.	salerec	Sales to receivables	Annual
57.	sgr	Sales growth	Annual
58.	sp	Sales to price	Annual
59.	stddolvol	Volatility of liquidity (dollar trading volume)	Monthly
60.	stdturn	Volatility of liquidity (share turnover)	Monthly
61.	stdacc	Accrual volatility	Quarterly
62.	stdcf	Cash flow volatility	Quarterly
63.	tb	Tax income to book income	Annual
64.	turn	Share turnover	Monthly

Appendix B: Vanishing Gradients

Vanishing gradients occur when gradients, which are used to update network weights through backpropagation, become extremely small, effectively preventing the network from further learning or making the learning process very slow. This problem arises because of the multiplication of gradients through layers. As the number of layers increases, the gradients of the network's previous layers tend to decrease exponentially, leading to minimal adjustments in these layers during training. This phenomenon makes models difficult to learn long-range dependencies or complex patterns in data.

The process of updating weight matric is via the following formula:

$$W - \alpha \frac{\partial L}{\partial W} \rightarrow W \tag{.1}$$

where W is the weight matric, α is the learning rate, L is the loss function.

For RNN, the updating process and gradient computation for weight W_{hh} is:

$$W_{hh} - \alpha \frac{\partial L_t}{\partial W_{hh}} \rightarrow W_{hh} \tag{.2}$$

$$\frac{\partial L_t}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_t}{}$$

equation

where L_t is the loss function at time step t, W_{hh} is the weights for the hidden state, h_t is the hidden state at time step t, h_k is the hidden state at an earlier time step k, $\frac{\partial h_t}{\partial h_k}$ is the gradient of the hidden state at time t with respect to the hidden state at time k.

Now we focus on the term $\frac{\partial h_t}{\partial h_k}$ from the computation of the gradient of the loss with respect to the weights at an earlier time step k.

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial h_{t-2}} \times \ldots \times \frac{\partial h_{k+1}}{\partial h_k} \tag{.4}$$

If we look at the gradient of the hidden state at time t with respect to the hidden state at time $t-1$, we have:

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial f\left(W_{hh} h_{t-1} + W_{xh} x_t + b_h\right)}{\partial h_{t-1}} = W_{hh} \times f'\left(W_{hh} h_{t-1} + W_{xh} x_t + b_h\right) \tag{.5}$$

where $f(\cdot)$ is the activation function. Usually, we will use sigmoid or the hyperbolic tangent as the activation function. However, we can notice $\frac{\partial h_t}{\partial h_{t-1}}$ is always smaller than 1[3].

Therefore, when we compute $\frac{\partial h_t}{\partial h_k}$ for a sequence, the product of many numbers less than 1 decreases exponentially. Mathematically, if $|W_{hh} \times f'| < 1$,

$$\left|\frac{\partial h_t}{\partial h_k}\right| = |W_{hh} \times f'|^n \tag{.6}$$

where n is the number of time steps between t and k, and this value can approach zero very quickly as n grows, which is the essence of the vanishing gradient problem. As a result, the gradients for weights associated with long-term dependencies become very small, meaning they contribute very little to the weight updates during training, or we can say that earlier "memory" are lost.

[3]When $f(\cdot)$ is sigmoid function $\sigma(z)$, the gradient $\sigma(z)'$ is $\sigma(z)(1-\sigma(z))$, which is maximally 0.25; when $f(\cdot)$ is tangent function $\tan(z)$, the gradient is $1-\tan(z)^2$, which is as most 1. Besides, the weights W_{hh} are usually initialized to values that keep the product within a reasonable range to prevent exploding gradients. Therefore, the product of W_{hh} and f' will be always smaller than 1.

Appendix C: LSTM and GRU

In LSTM, it has three gates: the input gate Γ_i, the forget gate Γ_f, and the output gate Γ_o and one cell state c_t, which is used to control the long-term memory.

The forget gate decides what information is discarded from the cell state. It looks at the previous hidden state h_{t-1} and the current input x_t, and outputs a number between 0 and 1 for each number in the cell state.

$$\Gamma_f = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{.7}$$

Similarly, an input gate is used to decide which new information to store in the cell state and the new information $\widetilde{c_t}$ is a function of a hidden state at the previous timestamp $t-1$ and input at timestamp t.

$$\Gamma_i = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \tag{.8}$$

$$\widetilde{c_t} = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \tag{.9}$$

After that, we can update the cell date c_t:

$$c_t = \Gamma_f \times c_{t-1} + \Gamma_i \times \widetilde{c_t} \tag{.10}$$

Finally, the LSTM needs to decide what to output. This output will be based on the cell state, but will be a filtered version. First, a sigmoid layer decides which parts of the cell state c_t to output. Then, the cell state is put through tanh (to push the values to be between -1 and 1) and multiplied by the output of the sigmoid gate, so that only the parts we decided to output are actually output.

$$\Gamma_o = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right) \tag{.11}$$

$$h_t = \Gamma_o \times \tanh\left(c_t\right) \tag{.12}$$

The LSTM's ability to add or remove information to the cell state, carefully regulated by the gates, allows it to handle dependencies in the input data.

Gated Recurrent Units (GRU) is a simplified version of LSTM. It has only two gates: update gate Γ_u and reset gate Γ_r.

The update gate Γ_u is to determine how much of the past information needs to be passed along to the future. It's similar to the LSTM's forget and input gates combined into one.

$$\Gamma_u = \sigma\left(W_u x_t + U_u h_{t-1} + b_u\right) \tag{.13}$$

The reset gate is used to decide how much of the past information to forget. It allows the model to decide how important each of the dimensions of the previous hidden state is.

$$\Gamma_r = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \tag{.14}$$

Utilizing the reset gate to determine which information to discard, the GRU creates a new memory content that will be used to make final decisions. The new memory content $\widetilde{h_t}$ which will use the reset gate to store the relevant information from the past.

$$\widetilde{h_t} = \tanh\left(W_h x_t + \Gamma_r \odot h_{t-1}\right) \tag{.15}$$

The hidden state is the final output of the GRU cell, which combines the old hidden state with the current memory content. The update gate directly influences how much of the past information (from previous time steps) needs to be passed along to the future.

$$h_t = \Gamma_u h_{t-1} + \left(1 - \Gamma_u\right) \widetilde{h_t} \tag{.16}$$

In summary, the GRU controls the flow of information like the LSTM, but without having to use a memory unit. It simply exposes the full hidden content without any control, but it regulates the information that is included or excluded from the state at each time step through the update and reset gates. This makes GRU easier to compute and train than LSTM, with some trade-offs in the capacity to model complex structures that might be better captured by LSTM. We employ both LSTM and GRU in our models.

Appendix D: Hyperparameter and Algorithm

According to Gu et al. (2020, 2021); Xuanling Yang and Zhu (2023), we summarize candidates in Table 1.

Table .8: Hyperparameter Candidates

Hyperparameter	Description	Candidates
λ	tuning parameter in l_1 penalty function	10^{-3}, 10^{-2}, 10^{-1}
α	learning rate in the Adam algorithm	10^{-4}, 10^{-3}, 10^{-2}
	number of neurons in beta hidden layer	8, 16, 64
M	batch size	64

We adopt the adaptive moment estimation algorithm (Adam), which is introduced by Kingma and Ba (2014). Adam computes adaptive learning rates for individual parameters using estimates of first and second moments of the gradients. At each step of training, a batch sent to the algorithm is randomly sampled from the training dataset. Algorithm 1 shows the Adam algorithm and Algorithm 2 show Batch Normalization process.

Algorithm 1 Adam algorithm

Require: α: Step size
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$ (Update parameters)
end while
return θ_t

Algorithm 2 Batch Normalization over mini-batch

Require: $\mathbf{X} = \{x_1, \ldots, x_m\}$: Input data (mini-batch)
Require: γ, β: Scale and shift parameters to be learned
Require: ϵ: A small constant for numerical stability

$\mu \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$ (Mini-batch mean)
$\sigma^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2$ (Mini-batch variance)
for each x_i in $\mathbf{X}$ **do**
    $\hat{x}_i \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$ (Normalize)
    $y_i \leftarrow \gamma \hat{x}_i + \beta$ (Scale and shift)
end for
return $\{y_1, \ldots, y_m\}$ (Batch-normalized output)

Appendix E: Robustness check

In this section, we check whether our best model SeqLAT has a robust performance with respect to the selection of assets in the training and testing samples. To be specific, we split all stocks into two groups with odd or even permnos, respectively. Accordingly, we divide the training and test sample into two subsamples. Then, we retrain our model with $K = 6$ on each sub-training sample and assess the out-of-sample performance on each sub-testing sample. Table 2 reports the out-of-sample results of total and predictive R^2's and long-short and long-only Sharpe ratios without transaction costs under four different scenarios. It can be seen that the performance of SeqLAT model is robust with respect to the sub-training and sub-testing samples, even when the sub-training and sub-testing samples are not the same.

Table .9: Statistical and Economic Performance for Robustness Check

Statistical performance

	Total R^2		Predictive R^2 (%)	
	Odd	Even	Odd	Even
Odd	17.45	18.32	1.32	1.45
Even	18.41	18.79	1.42	1.47

Economic performance

	Sharpe ratio for Long-short portfolios		Sharpe ratio for Long-only portfolios	
	Odd	Even	Odd	Even
Odd	2.90	2.87	1.83	1.95
Even	2.97	3.03	2.02	2.10