The effect of providing LSTM-based neural networks used for time-series predictions with relational data

Piotr Pilszczek[1]

[1] *Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND*

Abstract

Stock market forecasting is a challenging time-series prediction problem that has been widely studied using neural networks. Making predictions for just a single entity may not come as hard, but building a model that would forecast the closing market values for multiple stock companies can easily become challenging. To address this issue, I compare three proposed neural network architectural models: Basic Model (A) - a purely LSTM-based model with a separate sub-module for each stock company, Relational Model (B) - a LSTM-based model with relational attention mechanism and Graph Model (C) - a LSTM-based model with addition of a specific graph transformer layer. To quickly summarize all the results I obtained during tests: model (A) has always been the one with the greatest error value, while model (B) and model (C) have both been receiving similar results (and with lower error value than the model (A)) - but errors in the model (C) have always been slightly lower. It shows that one can easily improve neural networks accuracy by appending additional relational data regarding the given time-series. I explain the methodology and specific results later in the article.

Keywords

LSTM, Graphs, Neural Networks, RNN, Graph Transformers

1. Introduction

Modeling sequences has had a big breakthrough with the introduction of Long Short-Term Memory (LSTM) [1] neural networks - designed to solve problems like gradient explosions which occurred in the standard RNNs. LSTMs enable networks to capture long-range sequence dependencies and therefore became a dominant architecture used in time-series forecasting issues like predicting lost GPS signal [2], stock predictions [3] and earthquake forecasts [4]. Nevertheless, most models based on this architecture treat time sequences independently and as a result ignore relational structures such as stock dependencies within the same market sector.

Recently, more pressure has been put on graph-based neural networks [5], which could for example represent selected entities as their nodes, while also keeping track of relationships between them through parameterizing their edges [6]. Later additions such as attention mechanisms in graph convolutions (such as Transformer-based graph layers), further enhanced tracking important relationships [7].

The evolution I described motivates the idea behind this paper - researching the effects of extending LSTM-based models by appending attention-tracking mechanisms or graph transformer layers which would cause the neural networks to improve relationships' tackling.

*IVUS 2026: Information Society and University Studies 2026, May 12, Kaunas, Lithuania

pp310219@student.polsl.pl (P. Pilszczek)

Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-06

CEUR-WS.org/Vol-4317/paper10.pdf

2. Methodology

In this section I will briefly go through the notation used throughout this article. I denote the batch size as B, number of tickers (stock companies) as T, the length of the input time sequence as L. I denote graph as $G = (V, E)$ where V denotes the set of its vertices and E denotes the set of its edges. Because batching the input set does not influence the models' outcomes, I will also skip the notations of batched data when describing neural networks' architectures.

Firstly, the main criterion taken under consideration when comparing models' accuracy is the global error function - mean absolute percentage error - defined as follows:

$$\text{MAPE}(y, \hat{y}) = \frac{1}{T}\sum_{i=1}^{T}\left|\frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)} + \epsilon}\right|, \tag{1}$$

where the real expected output is denoted as: $y = \left(y^{(1)}, y^{(2)}, \dots, y^{(T)}\right) \in \mathbb{R}^T$ and the model's prediction vector is denoted as: $\hat{y} = \left(\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(T)}\right) \in \mathbb{R}^T$. I will reference this function as $\text{MAPE}(y, \hat{y})$. Additionally, in selected experiments later in the article, I also utilize the mean absolute error (MAE), defined as:

$$\text{MAE}(y, \hat{y}) = \frac{1}{T}\sum_{i=1}^{T}\left|y^{(i)} - \hat{y}^{(i)}\right|, \tag{2}$$

which provides an absolute-scale measure of prediction accuracy.
Another crucial part of training the models the same way will be a consistent and accurate loss function - which I found to be SmoothL1Loss [8] defined as follows:

$$\text{SmoothL1Loss}(y, \hat{y}) = \frac{1}{B}\sum_{i=1}^{B}\ell_\beta(\hat{y}_i - y_i), \tag{3}$$

where

$$\ell_\beta(r_i) = \begin{cases} \frac{1}{2\beta}r_i^2, |r_i| < \beta \\ |r_i| - \frac{\beta}{2}, \text{ otherwise} \end{cases} \tag{4}$$

I will reference this function as $\mathcal{L}(y, \hat{y}) = \text{SmoothL1Loss}(y, \hat{y})$.
Each neural network's training loop will use the same exact optimizer - Adam - with a given learning rate. To prevent overfitting while running the models through a learning loop, while also optimizing them as similarly as possible.

2.1. Basic Model (A)

The first model I use has been written purely for confirming the hypothesis - that's why I expect the error to be the highest one, compared to the two remaining models. The model treats all the tickers separately - the neural network sees no correlation between them, and therefore it can be treated as T different sub-models. The model is based purely on LSTM and does not contain any other advanced layers. In the architecture description I will denote the number of hidden features in the LSTM layer as H.

The network receives input data $X \in \mathbb{R}^{L \times T}$. It then separates the tensor $x^{(i)} \in \mathbb{R}^{L \times 1}$ for i-th ticker. Next, each tensor $x^{(i)}$ is being processed by its own LSTM layer into a new tensor:

$$\bar{z}^{(i)} = \text{LSTM}^{(i)}\left(x^{(i)}\right) = \left(h_1^{(i)}, h_2^{(i)}, \dots, h_L^{(i)}\right) \in \mathbb{R}^{L \times H} \tag{5}$$

The neural network picks the element $z^{(i)} = h_L^{(i)} \in \mathbb{R}^H$. In the next step, element $z^{(i)}$ is being processed by a fully connected (FC) linear layer:

$$\text{FC}^{(i)}(z) = W^{(i)} z + b^{(i)} \tag{6}$$

to obtain a prediction $\hat{y}^{(i)} = \text{FC}^{(i)}\left(z^{(i)}\right) \in \mathbb{R}$. As a result, we receive the final prediction:

$$\hat{y} = \left(\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(T)}\right) \in \mathbb{R}^T \tag{7}$$

2.2. Relational Model (B)

The Relational Model is the first neural network architecture that is supposed to show better results than the model (A). It does not see the tickers separately anymore - each ticker has its own sequence enriched with an embedding. All the representations are then merged by an attention mechanism that models out relations between the tickers. The prediction for one entity now depends on other predictions for different stock companies. I will denote the number of hidden features in the LSTM layer as H. I will denote the embedding dimension as E. I will denote the dimension of the query-key space as d_k.

The network receives input data $X \in \mathbb{R}^{L \times T}$. It then builds a data sequence for i-th ticker as follows. Firstly, I extend each time element $X_t \in \mathbb{R}^T$ with an embedding $e_i \in \mathbb{R}^E$ for the i-th ticker as $s_t^{(i)} = (X_t, e_i) \in \mathbb{R}^{T+E}$. I then build a sequence for that ticker $S^{(i)} = \left(s_1^{(i)}, s_2^{(i)}, \dots, s_L^{(i)}\right) \in \mathbb{R}^{L \times (T+E)}$. Next, I use the shared LSTM layer to process the sequence $S^{(i)}$ into a tensor:

$$\bar{z}^{(i)} = \text{LSTM}\left(S^{(i)}\right) = \left(h_1^{(i)}, h_2^{(i)}, \dots, h_L^{(i)}\right) \in \mathbb{R}^{L \times H} \tag{8}$$

The neural network picks the element $z^{(i)} = h_L^{(i)} \in \mathbb{R}^H$. In the next step, I build a projection of keys and queries used later in the attention mechanism $K^{(i)} = W_K z^{(i)} \in \mathbb{R}^{d_k}, Q^{(i)} = W_Q z^{(i)} \in \mathbb{R}^{d_k}$. I can now calculate the relation matrix for each ticker pair (i, j) :

$$A_{i,j} = \text{Softmax}\left(\frac{\left(Q^{(i)}\right)^T K^{(j)}}{\sqrt{d_k}}\right) \tag{9}$$

Now, I aggregate the results:

$$\tilde{z}^{(i)} = \sum_{j=1}^{T} A_{i,j} z^{(j)} \in \mathbb{R}^H \tag{10}$$

The model additionally trains the parameter $\alpha \in (0,1)$. It uses it to merge the original representation $z^{(i)}$ with the one that takes relations under consideration $-\tilde{z}^{(i)}$:

$$u^{(i)} = \alpha \tilde{z}^{(i)} + (1 - \alpha) z^{(i)} \tag{11}$$

Next, each vector is being normalized by LayerNorm and Dropout layers, ultimately providing the representation $v^{(i)}$ as $v^{(i)} = \text{Dropout}\left(\text{LayerNorm}\left(u^{(i)}\right)\right) \in \mathbb{R}^H$. The model can now predict the change of the data with the use of a MLP layer: $\Delta^{(i)} = \text{MLP}\left(v^{(i)}\right) \in \mathbb{R}$. Finally, the model provides the prediction for the i-th ticker:

$$\hat{y}^{(i)} = X_{L,i} + \Delta^{(i)}, \tag{12}$$

where $X_{L,i}$ is the last value from the input sequence for the i-th ticker. The output prediction can be formed as:

$$\hat{y} = \left(\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(T)}\right) \in \mathbb{R}^T \tag{13}$$

2.3. Graph Model (C)

The Graph Model is the second neural network architecture that is supposed to show better results than the model (A). It also does not see the tickers separately - the relations between stock companies are modeled by a convolutional graph layer TransformerConv [9], which is based on an attention mechanism for graphs. The model uses a complete graph between all the tickers - each ticker can influence another one on a graph level. The prediction for one entity now depends on other predictions for different stock companies. I will denote the number of hidden features in the LSTM layer as H. I will denote the size of the output sample from the TransformerConv layer as C.

The network receives input data $X \in \mathbb{R}^{L \times T}$. It then separates the tensor $x^{(i)} \in \mathbb{R}^{L \times 1}$ for i-th ticker. Next, each tensor $x^{(i)}$ is being processed by a LSTM layer into a new tensor: $\bar{z}^{(i)} = \text{LSTM}\left(x^{(i)}\right) = \left(h_1^{(i)}, h_2^{(i)}, \dots, h_L^{(i)}\right) \in \mathbb{R}^{L \times H}$.

The neural network picks the element $z^{(i)} = h_L^{(i)} \in \mathbb{R}^H$. In the next step, the model builds a complete graph $G = (V, E)$ with all the tickers, where $V = \{v_i : i = 1,2, \dots, T\}, E = \left\{\left(v_i, v_j\right) : i, j \in \{1,2, \dots, T\}, i \neq j\right\}$. Now, the neural network processes the tickers' representations using the TransformerConv layer based on an architecture from article [9]. It obtains the following representation $H_{\text{TC}} = \left(h_{\text{G}}^{(1)}, h_{\text{G}}^{(2)}, \dots, h_{\text{G}}^{(T)}\right) \in \mathbb{R}^{T \times C}$. The model can now use a linear fully connected (FC) layer to receive a change value:

$$\Delta^{(i)} = W_{\text{FC}} h_{\text{G}}^{(i)} + b_{\text{FC}} \in \mathbb{R} \tag{14}$$

Finally, the neural network makes its final prediction sequence:

$$\hat{y}^{(i)} = X_{L,i} + \Delta^{(i)}, \tag{15}$$

$$\hat{y} = \left(\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(T)}\right) \in \mathbb{R}^T. \tag{16}$$

A very important part of this model is graph freezing - on a freeze_epoch epoch the layer completely stops learning which prevents troublesome side effects such as overfitting.

3. Experiments

3.1. Dataset

The dataset comes from the website Yahoo Finance [10]. I use 12 different tickers, grouped into 3 categories: technologies (Apple, Microsoft, Google, Amazon), semiconductors (TSMC, Broadcom, Intel, Qualcomm) and finances (JPMorgan Chase, Bank of America, Citigroup, The Goldman Sachs Group). This choice has been made to make it easier for the neural networks to incorporate relations between the given stock companies.

The specific data used in this project comes from the dates: 2020-01-01-2025-12-01. I split the dataset into two parts: training (80% of the entire dataset) and test (20% of the entire dataset). Finally, to obtain the set used in the project I merge the datasets using the dates shared by all subsets (each subset matching each ticker).

3.2. Basic Model (A)

The Basic Model serves as a baseline architecture. As discussed in Section (2), this model treats each ticker independently - therefore, it is expected to achieve the highest error.

Training setup. The model can be interpreted as a multi-ticker LSTM architecture which consists of T separate sub-modules - one per ticker. I set the hidden features hyperparameter to $H = 32$ (inside of a single LSTM recurrent layer). The model uses learning rate $\eta = 4 \times 10^{-4}$.

The objective during training is to minimize the SmoothL1Loss loss function introduced in Section (2). I train the model for maximum 1000 epochs, although the process has always terminated earlier thanks to the early stopping mechanism with patience equal to 20 epochs and a minimum improvement threshold of 10^{-5} on the test loss. Once early stopping has been triggered, all the model's best parameters will be restored.

Results and observations. Figure (1) illustrates the global MAE error over the entire test time period. As expected, the model fits the expected values relatively well, but its final error remains consistently higher than those obtained by the relational and graph-based models.

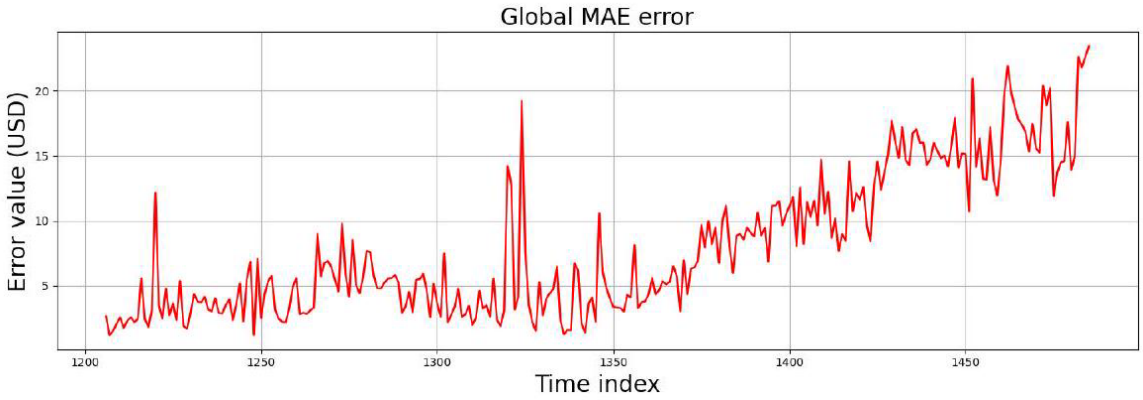


Figure 1: Test MAE errors for the Basic Model (A) throughout the test time

We can obviously observe a behavior that confirms the limitations of the chosen approach while the LSTM layers are capable of capturing dependencies over short or repeated periods of time, the absence of cross-ticker relationship handling makes the model ignore very crucial correlations between related stocks.

Overall, the model provides an important and strong reference point for evaluation of benefits introduced by appending relational mechanisms into models (B) and (C). In terms of the global evaluation metric, the Basic Model achieves a mean absolute percentage error **2.71**%, which

serves as a reference baseline for assessing the improvements introduced by relational and graph-based architectures.

3.3. Relational Model (B)

The Relational Model extends the baseline architecture by explicitly modeling dependencies between different tickers using an attention-based relational mechanism. Unlike the Basic Model (A), this architecture allows each ticker's prediction to be influenced by representations learned for other tickers, enabling the model to exploit correlations present in the data. As a result, it is expected to achieve lower error values than the baseline approach.

Training setup. I implemented the Relational Model as an architecture that utilizes shared LSTM layers combined with a learnable embeddings mechanism as well as an attention addition. I set the LSTM hidden features amount to $H = 32$ (inside of a single LSTM recurrent layer). I also set the Query-Key dimension size to $d_k = 32$. The model uses learning rate $\eta = 3 \times 10^{-4}$.

The objective during training is to minimize the SmoothL1Loss loss function introduced in Section (2) combined with an entropy-based regulator. I train the model for maximum 1000 epochs, although the process has always terminated earlier thanks to the early stopping mechanism with patience equal to 40 epochs - and a minimum improvement threshold of 10^{-5} on the test loss. Once early stopping has been triggered, all the model's best parameters will be restored.

Results and observations. Figure (2) illustrates the global MAE error over the entire test time period. As expected, compared to the Basic Model (A), a consistent reduction in error can be observed - that's why we can conclude that incorporating relational information between tickers improved predictive accuracy by a significant margin.

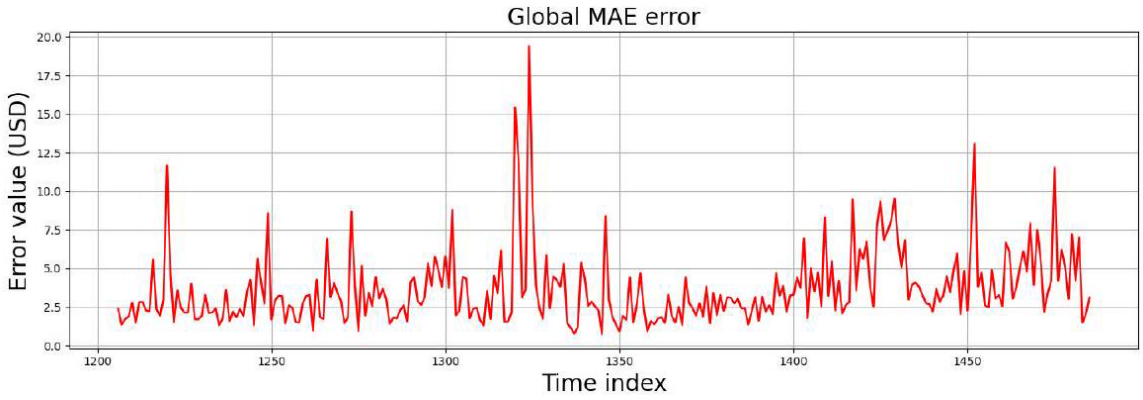


Figure 2: Test MAE errors for the Relational Model (B) throughout the test time

The relational attention mechanism enabled the model to dynamically adjust tickers' influence onto its predictions, allowing it to better capture market movements and stock dependencies.

Overall, the Relational Model can easily demonstrate its advantages, showing that incorporating relationship information leads to a noticeable improvement over purely LSTM-based modeling of independent time-series. When evaluated using the global metric, the Relational Model attains a mean absolute percentage error of **1.72%**, demonstrating a substantial improvement over the baseline by effectively incorporating inter-ticker relational information.

3.4. Graph Model (C)

The Graph Model represents the most expressive architecture considered in this study, as it incorporates relational information using an explicit graph-based layer. Unlike the Relational Model (B), where dependencies are learned implicitly through attention over embedding representations, this approach models interactions between tickers directly on a graph structure. As a result, it is expected to provide the most accurate predictions among the evaluated models.

Training setup. I implemented this model to utilize LSTM layer followed by a TransformerConv graph layer [9] which operates on a complete graph $G = (V, E)$ over all the tickers. I set the LSTM hidden features amount to $H = 16$ (inside of a single LSTM recurrent layer). I set the graph output dimension to $C = 64$ with the use of 2 attention heads (number of objects performing attention module computations). The model uses learning rate $\eta = 4 \times 10^{-4}$.

The objective during training is to minimize the SmoothL1Loss loss function introduced in Section (2). I train the model for maximum 1000 epochs, although the process has always terminated earlier thanks to the early stopping mechanism with patience equal to 40 epochs - and a minimum improvement threshold of 10^{-5} on the test loss. Similarly to both previous learning mechanisms, once early stopping has been triggered, all the model's best parameters will be restored.

To stabilize training and prevent overfitting of the graph attention mechanism, the parameters of the TransformerConv layer are frozen after a predefined number of epochs - specifically, after FREEZE_EPOCH $= 15$, all parameters of the graph convolution layer are excluded from further calculations and updates.

Results and observations. Figure (3) illustrates the global MAE error over the entire test time period. Again, as expected, compared to both the Basic Model (A) and the Relational Model (B), this architecture consistently achieves the lowest error values.

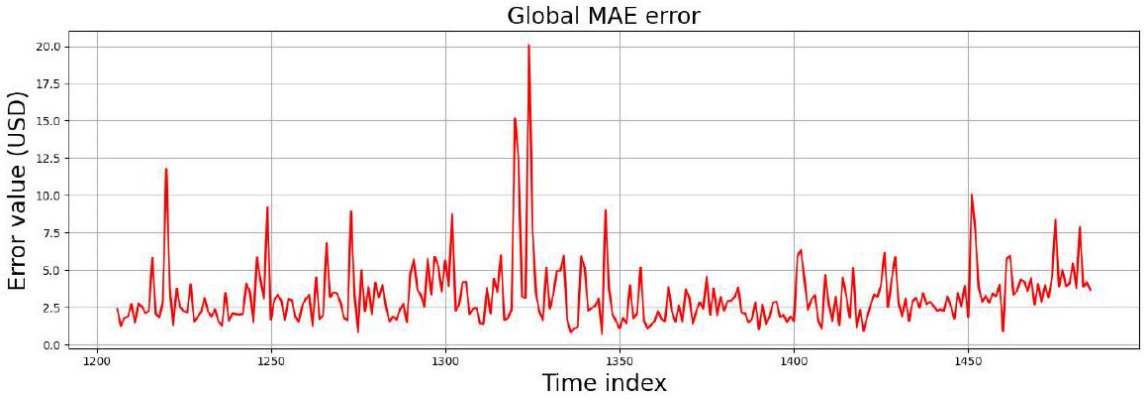


Figure 3: Test MAE errors for the Graph Model (C) throughout the test time

The model formulation which has been based on a explicit complete graph architecture allows the model to pass relationship values throughout the learning process - it then leads to the network capturing dependencies other models would have always ignored. The graph freezing strategy also allows the model to stop overfitting and makes the adaptation more accurate and meaningful.

Overall, the Graph Model demonstrates that combining temporal LSTM-based time-series modeling with explicit graph-based relational reasoning results in the most accurate predictions

among all the approaches I have researched. Using the mean absolute percentage error as the global evaluation metric, the Graph Model achieves a **1.56** % MAPE, which is the lowest among all evaluated architectures and confirms the effectiveness of explicit graph-based relational modeling.

3.5. Prediction analysis

A great complement of the error metrics I wrote about in this section would be to provide a qualitative analysis of the actual output of neural networks created for this project. In this part forecasts for a representative ticker (JPMorgan Chase) will be discussed. The figure (4) below is an illustration of the behaviors of all three models.



(a) Basic Model (A) (b) Relational Model (B) (c) Graph Model (C)

Figure 4: Test-set stock price predictions for JPMorgan Chase. The blue curve denotes original prices, orange curves represent model predictions, and dashed lines mark training/test boundaries and the forecast horizon.

Discussion. Figure 4 reveals clear differences between the three architectures that are not fully captured by global error metrics alone.
The Basic Model (A) provides relatively accurate time-series predictions. However, the prediction errors tend to grow relatively quickly compared to other models I described. This behavior may be visible particularly well in the later parts of the test interval. The obvious cause of this issue lies in the network's architecture - treating ticker independently stops the model from improving itself over a prolonged time horizon.

The Relational Model (B) on the other hand demonstrates a totally different error profile. Instead of none-to-small errors close to the start and big error values while approaching the end of the test period, this model achieves small but constant inaccuracies throughout the entire test time. This pattern shows that utilizing relational data in a attention-based mechanism can successfully mitigate large deviations - however it will not get rid of local inaccuracies. The root of this outcome once again lies in the architecture - this time however, it is the result of blending temporal representations with attention information.

The Graph Model (C) shows the strongest performance. Its predictions track the original test data highly accurately, providing almost no deviation. Its stability is the result of formulating very explicit and powerful graph layer that propagates relationship data throughout the network using the TransformerConv layer. By modeling the relations on a graph, the model can track very subtle correlations not visible to other networks' architectures. Furthermore, the freezing of the graph layer stabilizes learned relational patterns even stronger preventing model overfitting.

Importantly, the error patterns I just described are not limited to this specific ticker - the same behavior may be observed in all other evaluated time-series, which confirms the conclusions.

3.6. Comparison

The entire comparison of all the architectures and parameters can bee seen in the tables below. The comparison of all the MAPE and MAE errors can be seen in the Table (2).

Table 1
Comparison of model architectures and training configurations

Property	Basic Model (A)	Relational Model (B)	Graph Model (C)
LSTM hidden features (H)	32	32	16
LSTM layers	1	1	1
Ticker embedding dimension	-	$E = 8$	-
Relational mechanism	None	Attention	TransformerConv
Query-Key dimension (d_k)	-	32	-
Graph output size (C)	-	-	64
Attention heads	-	1	2
Dropout parameter	0	0.05	0.05
Loss function	SmoothL1Loss	SmoothL1Loss + entropy	SmoothL1Loss
Learning rate (η)	4×10^{-4}	3×10^{-4}	4×10^{-4}
Early stopping patience	20	40	20
Graph freezing	-	-	Yes (epoch 15)

Table 2
Final test performance comparison of evaluated models. Each metric is calculated as mean of the metric over the test period.

Model	Mean test MAE	Mean test MAPE
Basic Model (A)	7.96 USD	2.71%
Relational Model (B)	3.64 USD	1.72%
Graph Model (C)	3.26 USD	1.56%

4. Conclusion

In this paper I investigated the impact of incorporating relational information into LSTM-based neural networks. Three architectures of increasing correlations were compared: a purely LSTM-based model (Basic Model (A)), an attention-based relational model (Relational Model (B)), and a graph-based architecture utilizing a Transformer-style graph convolution (Graph Model (C)).

The results of my experiments have stayed consistent with the hypothesis of this research explicitly modeling relationships between time-series entities leads to a measurable improvement in accuracy of the forecasts. The more relational data the model has, the greater are its improvement metrics.

The Basic Model's (A) results are significantly less accurate then the ones obtained from remaining architectures - though it provides a valuable base for comparison. Introducing relational data via attention mechanism in the Relational Model (B) reduces error values by a huge margin. The best results were obtained with the Graph Model (C), which explicitly encodes ticker relations using a TransformerConv graph transformer layer.

Overall, the findings suggest that appending relational data into neural network's architecture is a crucial component of models supposed to solve multi-time-series forecasting problems. Making such improvement is not restricted to a single pattern - however research in this paper shows that attention mechanism can be slightly less efficient compared to a graph-based solution.

5. Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (1997) 1735-1780.
[2] P. Zerdzinski, D. Połap, N. Wawrzyniak, G. Zaniewicz, A. Zielonka, Attention-enhanced temporal lstm network as a sonar position predictor for side-scanning sonar waterfall image, IEEE Access (2025).
[3] S. Sang, L. Li, A novel variant of lstm stock prediction method incorporating attention mechanism, Mathematics 12 (2024) 945.
[4] W. Plechta, Comparison of neural network models with gru and lstm layer for earthquake prediction., in: IVUS, 2024, pp. 89-98.
[5] J. Xu, J. Yang, Y. Huang, L. Y. Por, X. Chen, C. Zhao, Dfgnet: A dual-pathway graph neural network via frequency decomposition for spatiotemporal forecasting, Expert Systems with Applications 297 (2026) 129518.
[6] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (2009) 61-80.
[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, 2018. URL: https://arxiv.org/abs/1710.10903.
[8] PyTorch, Pytorch documentation, https://docs.pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html, 2025. Accessed: 2025-12-12.
[9] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, Y. Sun, Masked label prediction: Unified message passing model for semi-supervised classification, 2021. URL: https://arxiv.org/abs/2009.03509. arXiv:2009.03509.

[10] Yahoo, Yahoo finance: Historical market data, https://finance.yahoo.com/, 2025. Accessed: 2025-12-12.