# A SURVEY ON GRAPH NEURAL NETWORKS FOR PRICE PREDICTION

Yifei LIU
*Department of Computer Science*
*University of Auckland*
Auckland, New Zealand
yliu523@aucklanduni.ac.nz

Kwong Chun WU
*Department of Computer Science*
*University of Auckland*
Auckland, New Zealand
kwu764@aucklanduni.ac.nz

Kam leong AO
*Department of Computer Science*
*University of Auckland*
Auckland, New Zealand
kao323@aucklanduni.ac.nz

Yong YU
*Department of Computer Science*
*University of Auckland*
Auckland, New Zealand
yyu482@aucklanduni.ac.nz

Mark CHEN
*Department of Computer Science*
*University of Auckland*
Auckland, New Zealand
zche677@aucklanduni.ac.nz

***Abstract – Price prediction has been one of the most discussed topics in both statistical and financial fields. Numerous studies have been carried out to predict the stock market especially the in past decades. In view of the growing capitalization and penetration of cryptocurrency trading, more researches have been conducted on the price prediction of this emerging sector in recent years. In this literature review, we would like to discuss the feasibility of applying bleeding-edge machine learning techniques to the task of commonly-traded price prediction with graph attention networks being the main focus of our study. We break down the strengths and the weaknesses of each graph-based algorithm as we study a combination of models most suitable for this task. Additionally, we referred a recent solution for speeding up the matrix computation which would be beneficial in our research as a future consideration.***

***Keywords – Price prediction, cryptocurrency, neural network, graph attention network***

## I. INTRODUCTION

Since the first cryptocurrency, Bitcoin, was born in 2008, the cryptocurrency trading has been a popular investment in the last decade. According to Statistia [8], there are nearly 6000 cryptocurrencies available in the market in 2021, which is ten times compared with 5 years ago. At the same period, the capitalization grew 100 times, from 18000 billion US dollars in 2016 to a recent peak 1.8M billion US dollars in May 2021.

Same as the stock market in the past decades, many researches were carried out for price prediction of the cryptocurrencies in recent years. One of the reasons is that the volatility of cryptocurrencies is observed stronger than usual stock market. An over 10% daily price movement is very common in the cryptocurrency market. Taking Bitcoin as an example, the price of Bitcoin was US$ 952 at the end of 2016. It reached its all-time high US$ 64854 on 14 April 2021. The price of Bitcoin roamed between US$ 28000 and US$ 53000 from then on. Even the largest capitalized cryptocurrency has such a high volatility, some lower tiers cryptocurrencies could rise over 10 times or fall more than 90% in a short period of time. Therefore, an effective price prediction could be a large room of profit which motivated many researchers to study the topic.

## II. BACKGROUND

At the moment, most of the researches focused on the Bitcoin due to its highest market value and the longest history among all the cryptocurrencies. Among the researches, one of the most studied scope was the trend prediction, which meant predicting whether the price would go upward or downward. Ladislav Kristoufek [13] found that there was a positive correlation between the price level of Bitcoin and the keywords searched in Google Trends and Wikipedia. Polasik et. al [17] performed sentiment analysis on the price of Bitcoin and the popularity of cryptocurrency in medias, which they concluded the price of Bitcoin was pushed by its popularity.

However, investors may wish to have a more detailed and accurate information to make decisions, only the price movement direction was obviously inadequate. For example, without a price prediction, investors could not perform risk analysis which is a critical part in cryptocurrency investment due to its volatile nature.

In light of the continuous trading pattern of the cryptocurrency market, it provided a good platform to apply time series model on the data. However, according to the study by Muhammad J Amjad et al. [1], several limitations were found in the ARIMA model application on cryptocurrency data, such as a lack of probabilistic interpretation. In addition, the predictability of ARIMA model in their research was undesirable for investment.

Cryptocurrency is a virtual commodity without any real substances to back up its value nor generate any additional value. Pavel Ciaian et al. [6] discovered that supply and demand is one of the key factors to impact the price of Bitcoin. Same as other investments, investors need to consider the opportunity cost when allocating their funds, especially when they invest on such virtual asset. Inspired by the study performed by Pratik Patil et al.[16] and Dennys C.A.Mallqui [15], we agreed that there may be a correlation between the price of cryptocurrency and some indexes, currencies, price of stocks or commodities. Therefore, we are proposing to use graph neural network methods to predict the price of cryptocurrencies in our study.

## III. Graph Neural Network

Graph Neural Network (GNN) has become more popular in applications in recent years. The algorithm was successfully applied in recommender systems and social network fields. GNN could solve the problems in dealing with graphical structure or random size graphs, while Convolutional Neural Network (CNN) could not perform decently beyond Euclidean data (2-D images and 1-D texts) [9], thus, GNN is one of the effective methods to deal with dynamic graph structures. The graph structure behind the algorithm is able to connect considerable numbers of items by edges. Two nodes connected by an edge mean that they have similar embeddings. However, if the value of an embedding changes, the graph structure is difficult to remain the same as the original version that this issue would be discussed later.

One significant feature that could differentiate convolution operators and recurrent operators is that: the edges of graph structure under the former one have different weights, but the latter one's edges have the same weights [9]. In other words, the graph structure in GNN shares the same weight of each edge, but the graph in GAT contains different weights. That is one of the reasons why GAT performs more efficiently in analyzing sequence-related assignments. For the graph behind the GNN, there exists a state embedding $h_v \in \mathbb{R}^s$, which is a s-dimension vector of node $v$, comprises the neighbours' and its own message to operate message-passing, also $o_v$ named the output of node $v$ is produced by the state embedding and $x_v$ named the feature of $v$. Hence $h_v$ and $o_v$ are defined as :

$$h_v = f(x_v, x_{co[v]}, h_{ne_v}, x_{ne_v})$$

$$o_v = g(h_v, x_v)$$

where $f$ is the local transition function that is a parametric function, and $g$ is the local output function that the process of producing output is described, $x_{co[v]}$ is the edge features connected to $v$ , $h_{ne_v}$ is the state embedding of the neighbourhood set of node $v$, $x_{ne_v}$ is the features of the neighbourhood set of node $v$.

Therefore, in terms of GNN application achievements, Jin W. et al. [12] constructed a social recommendation system with GraphRec, which is based on the GNN framework to joint three different graphs at the same time for the sake of seeking more accurate social recommendation system. They generated three graphs namely user-user social relation graph, item-opinion graph, and item-opinion graph where the first two graphs concatenated as the user modelling and the last one was the item modelling which those graphs were built depending on individual's interactions in social media, the preferences of browsing item in applications, and the rating or comments toward specific items. The paper provided an idea of the capability of dealing with multiple graphs simultaneously by neural networks. Such technique was not only reducing the time of handling different graphs separately, but also delivered a more accurate result on analysis and prediction. On the other hand, according to a study [14], Y Ma et al. proposed a method called Dynamic Graph Neural Network (DyGNN) to solve an issue of passively updating the graph structure; this technique could pursue an updating graph structure in real time.

Although the robustness and versatility made GNN to be more popular, some limitations were found in the algorithm. The performance of GNN algorithm highly relies on the graph structure so that the vulnerability of graph structure would be one of the biggest issues. A graph structure may perform inefficiently as long as some noises were contained in the graph, the model operation would be interfered when the embeddings with maliciously altered. Jin et al. [12] summed up possible methods to attack the graph structure. For GNN, the types of perturbation can be divided into adding or deleting edges, modifying edges, and nodes injection, also there were eight various methods to maliciously modify the model. In addition, they introduced a number of methods to fight back those adversarial attacks, such as graph purification and adversarial training.

The above limitations lead us to to look further into Graph Attention Network.

## IV. Graph Attention Network

The underlying data representation of Convolutional Neural Network has one major limitation which the data must be a grid-like data structure. In order to solve the problem, graph attention network [19] is introduced. The idea of graph attention network is based on the combination of general graph neural network and attention mechanisms introduced in [18]. The basic idea of the graph attention network is generalized by the transformer's attention mechanism [18]. Graph attention network [19] is a message-passing network that every node iteratively updates its feature vector by a shared weight matrix which is a linear transformation of a feature vector and, most importantly, attention mechanism allows every node to compute attention coefficient for its neighbour and eventually forms a weighted adjacency matrix.

### *A. Learnable Linear Transformer: Shared Weight Matrix*

The input of our graph attention network would be a set of nodes with corresponding feature vectors,

$$h = \{\vec{h}_1, \vec{h}_2, \dots, \vec{h}_N\}, \qquad \vec{h}_i \in \mathbb{R}^F$$

where N is the number of nodes and F is the number of features in each node. At the beginning of the layer, graph attention network performs exactly the same as a general graph neural network by multiplying a shared learnable weight matrix $\boldsymbol{W}$ to linearly transform the inputted feature vectors for each node. This shared linear transformation $\boldsymbol{W}$ enables us to obtain enough expressive power to transform the input feature vectors into higher levels.

### *B. Apply Attention Mechanism on GAT*

The major difference between general graph neural network and graph attention network is the attention mechanism. This mechanism allows graph attention network to learn the importance of its neighbours apart from shared weight matrix **W** in general graph neural network. It learns the attention coefficient $\alpha$ which determines the significance between node-neighbour pairs. There are lots of different attention mechanisms to calculate the attention coefficients. In [19], they used a shared single layer neural network to calculate the attention coefficient $\alpha$ for each node-neighbour pair. The input of this network is two transformed node feature vectors, one is the transformed feature vector of the query node itself and another one is the transformed feature vector of the neighbour node. The output of this network would be the importance between this query node and its neighbours, namely, the attention coefficient $\alpha$ of query node-neighbour pair. The formula for attention coefficient $\alpha$ computation is as below:

$$\alpha_{ij} = \frac{\exp\left(LeakyReLU\left(\vec{a}^T\left[\boldsymbol{W}\vec{h}_i \parallel \boldsymbol{W}\vec{h}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(LeakyReLU\left(\vec{a}^T\left[\boldsymbol{W}\vec{h}_i \parallel \boldsymbol{W}\vec{h}_k\right]\right)\right)}$$

where $T$ represents a transposition, $\parallel$ represents concatenation and $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is some neighborhood of node $i$ in the graph.

We usually use the softmax function to normalize the attention coefficient $\alpha$ to make sure the value of attention coefficient for a particular node and all its neighbours sums up to 1. The softmax function is also applied as shown above in the formula.

In order to include the attention mechanism in the graph neural network and make it as a graph attention network, we would multiply the transformed feature vectors $\boldsymbol{W}\vec{h}$ of the query node's neighbours with the corresponding normalized attention coefficient $\alpha$ and aggregate them together to obtain $\vec{h}_i'$ when we are updating the query node's feature vector. As a result of that procedure, the important neighbours of that query node are amplified and the less important neighbours of that node are suppressed. The equation in [19] to aggregate transformed feature vectors $\boldsymbol{W}\vec{h}$ with normalized attention coefficient $\alpha$ is shown as below.

$$\vec{h}_i' = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\boldsymbol{W}\vec{h}_j\right), where\ \sigma\ implies\ nonlinearity$$

### C. Apply Multi-head Attention Mechanism on GAT

In order to stabilize the learning process of self-attention, the multi-head attention mechanism was also introduced in [19], which means $K$ independent attention mechanisms are applied instead of 1 attention mechanism mentioned above. Before the final layer, or before the prediction, the result of each different attention mechanism would be concatenated together which means $KF'$ features are outputted here instead of $1F'$ features mentioned above. The formula in [19] to aggregate transformed feature vectors $\boldsymbol{W}\vec{h}$ with normalized attention coefficient $\alpha$ before the prediction layer is shown below.

$$\vec{h}_i' = \Big\Vert\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\boldsymbol{W}^k\vec{h}_j\right), for\ k = 1\ to\ K$$

where || represents concatenation, $\alpha_{ij}^k$are normalized attention coefficients from the k-th attention mechanism $\alpha^k$.

However, it is clearly not reasonable to output $KF'$ concatenated features when it comes to the final layer of the network , namely the prediction layer. In [19], averaging is employed to transform the $KF'$ concatenated features into F' features. The formula in [19] to aggregate transformed feature vectors with normalized attention coefficient in the prediction layer which performs averaging is shown below.

$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\boldsymbol{W}^k\vec{h}_j\right)$$

Thus, apart from the shared weight matrix which is a learnable linear transformation learned in general graph neural network, by including attention mechanism, graph attention network also learns a weighted adjacency matrix that contains all the normalized attention coefficients which measures the significance of the neighbours for query node.

### D. Limitations of GAT

Nevertheless, graph attention networks also have a few limitations. Standard graph attention network can only compute static attention rather than dynamic attention [5] which means the ranking of attention coefficients is globally for all nodes in the graph regardless of the query node itself. In the original graph attention network [19] introduced above, the learned weighted matrix and the learned attention coefficient adjacency matrix are applied consecutively which result in collapsing into a single linear layer which leads to static attention. This limitation can be overcome by applying the learned weighted adjacency matrix after the nonlinearly and the learned weighted matrix after the concatenation [5]. Such a change of the operation would ensure the graph attention networks be able to produce a dynamic attention rather than a static attention. The modified formula version of graph attention network version 2 compared with original version of graph attention network is shown below.

GAT(Veličković et al., 2018):

$$e\left(h_i, h_j\right) = LeakyReLU(\boldsymbol{a}^T \cdot \left[\boldsymbol{W}h_i \parallel \boldsymbol{W}h_j\right])$$

GATv2(Brody, S., 2021):

$$e\left(h_i, h_j\right) = \boldsymbol{a}^T LeakyReLU(\boldsymbol{W} \cdot \left[h_i \parallel h_j\right])$$

In addition, Graph attention network itself is not time-aware which means it can only model graph structure data rather than time series data. Our topic here is essentially cryptocurrency and stock market price prediction, which means the data structure we are dealing with would have to be Multivariate Time Series. While a graph attention network can only model the data for a specific date, clearly, using a graph attention network alone cannot solve our problem.

## V. TIME SENSITIVITY OF GAT

We have studied 3 graph attention network models for processing time series data. They all used the recurrent neural network to get time series information and the graph attention network to find the relationship between nodes in graph data. Therefore, the two models can be used alternatively to get the best result.

The following sections of this section will have a focus on the of use graph neural networks to process time series data. In detail, would like to explore how we can effectively combine graph attention network and recurrent neural network for the best possible results. Here we have found two studies about the prediction of traffic conditions in different periods and regions. They both used GNN and RNN and combined these two to generate a new model.

### A. Graph Attention Recurrent Neural Networks [7]

The first model is called graph attention recurrent neural networks (GA-RNNS). The data is the traffic flow at different times in different locations. Firstly, the total data is divided into two parts by a time $T_a$. Then the data before $T_a$ is equally divided into $L$ parts as historical data. The data after $T_a$ is equally divided into $P$ parts as the future predict target data. Secondly, transforming the data into a graph structure $G = (V, E)$. Each traffic location in the data is a vertex, and the edge is determined by the spatial network or absolute distance

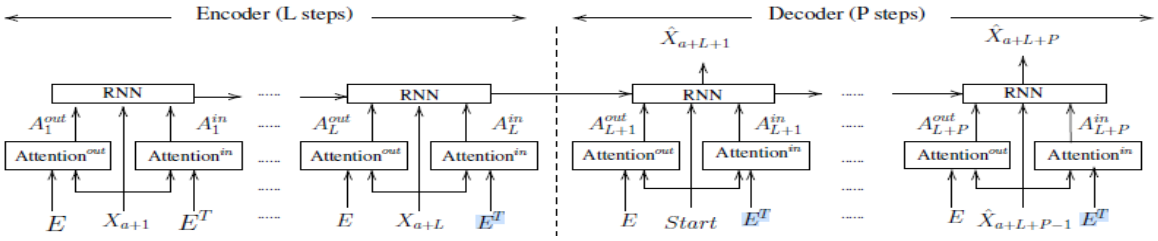


Fig.1 Graph Attention Recurrent Neural Network [7]

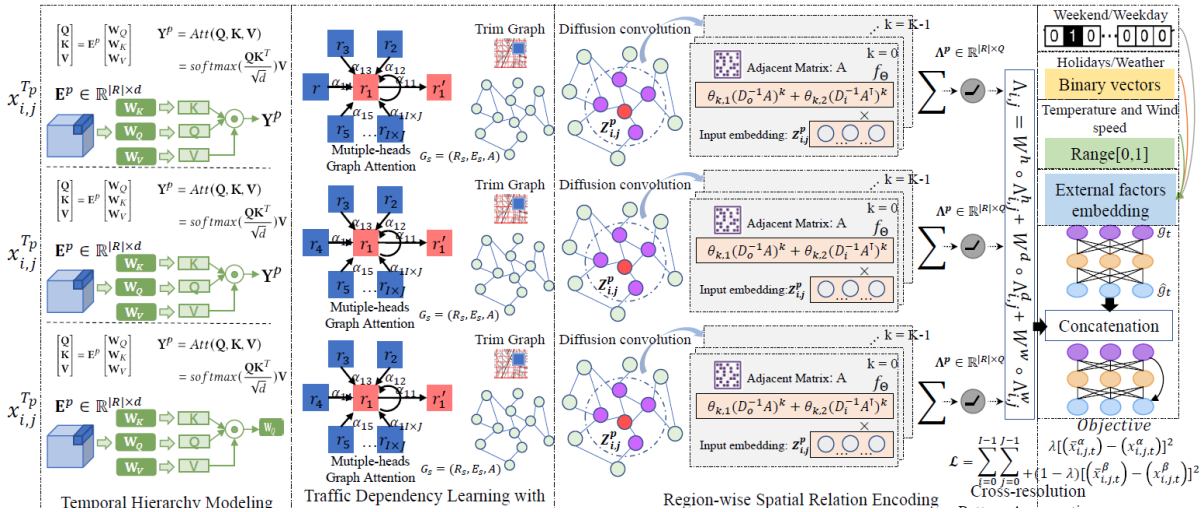


Fig.2 Spatial-Temporal Graph Diffusion Network [20]

between the locations. The feature contains the speed and traffic flow. Finally, the model uses the above data, inputting $L$ historical graph data for training, and using $P$ future graph data for prediction.

GA-RNNS consists of two parts, as shown in figure 1. The first part is the attention part. For the graph data of different regions and periods, it uses the incoming and outgoing traffic data to obtain two attention weight matrices. The second part is the RNN part. Putting the same period's input and output weight matrices into the RNN unit together with the graph data, the result is regionally related and time related. As shown in the figure 1, the left half is the historical data of the $L$ step, and the right half is the future prediction of the $P$ step. $X$ refers to the graph data of a specific period, $E$ and $E^T$ are the weight matrices of the output and input traffic, respectively. $A^{out}$ and $A^{in}$ are two attention matrices obtained through attention, corresponding to the output and input traffic conditions, respectively.

Before importing data to RNN, using diffusion convolution for graph data $X$, $A^{out}$ and $A^{in}$ to get a matrix contains all the features. Finally, input the matrix of diffusion convolution into RNN, where Gated Recurrent Unit is used as the unit of RNN.

### B. *Spatial-Temporal Graph Diffusion Network* [20]

The second model is called Spatial-Temporal Graph Diffusion Network (ST-GDN). Similar to the first model, it also predicts the traffic flow through the input and output traffic flow in different regions and times. The model structure is in figure 2.

Firstly, we define a temporal resolution $p$, which contains hourly, daily and weekly dimensions. We divide all traffic flow data $X$ into data $X_{i,j}^{T_p}$ according to regional coordinates $i, j$, and time interval $T_p$. Then we perform Temporal Hierarchy Modelling on these data to get $Y^p$, which is the learned resolution-aware hidden representation of all regions. Taking $Y^p$ of each region as the feature to create the graph data and using Graph Attention Network to get the aggregated feature embedding $Z_{i,j}^p$. As we know, the data has been separated into $L$ parts based on the temporal resolution $p$. By using the high-order relation modelling to propagate and aggregate information from the $l$-th layer of $Z_{i,j}^p$ to the ($l$+1)-th layer, we would finally get a $Z_{i,j}^p$ with features of all the layers. Based on the different temporal resolution $p$, getting three resolution-aware traffic representations could enable them to obtain the conclusive multi-resolution traffic representation for different regions.

Moreover, the two models are similar in the method of using GAT and RNN. We generate a time series graph structure data by different time intervals and apply GAT to get hidden relations for each graph. We could acquire the result by inputting these different time series graph data into RNN.

### C. *Financial Graph Attention Network* [10]

The price fluctuation of the market is much more complicated than the traffic flow even though the market of commodities is very similar to the traffic data structure. Therefore we also referred to a model specifically for stocks which is called Financial Graph Attention Network (FinGAT) as shown in figure 3.

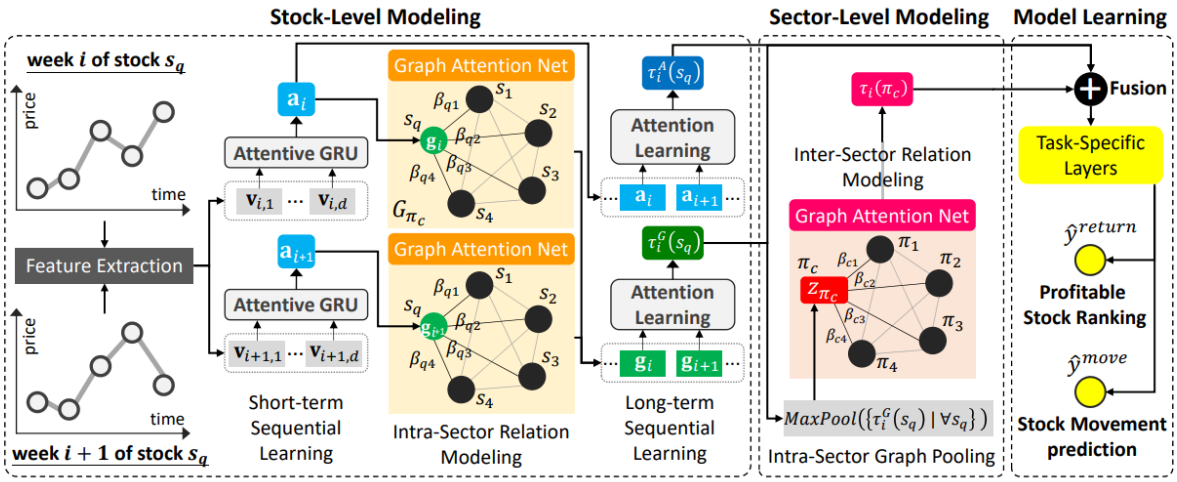


Fig.3 Financial Graph Attention Network [10]

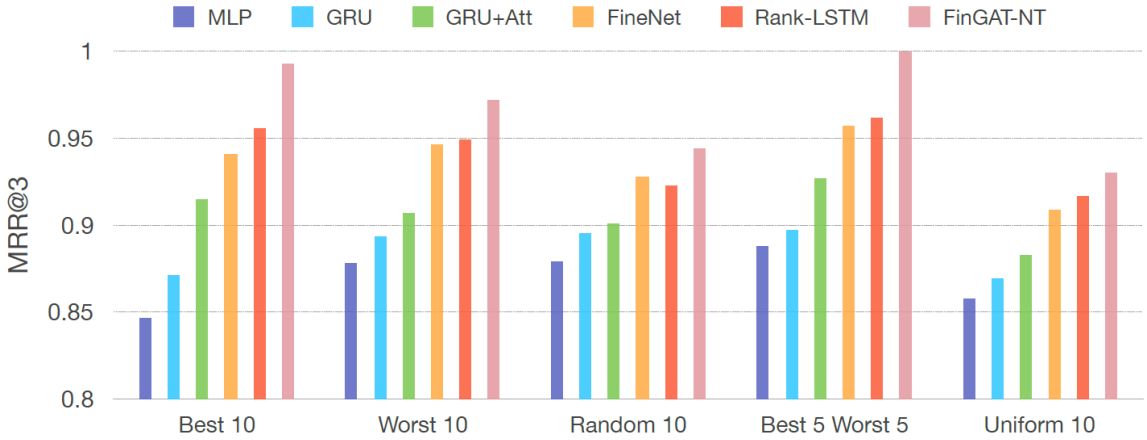


Fig.4 Results on recommendation of MRR@3 for different data subsets of Taiwan Stock dataset. [10]

The source data is divided by weekly intervals so that each piece of data having five days of transaction records. By performing Short-term Sequential Learning on these five days of data, with the use of Gated Recurrent Unit and feed-forward neural network-based attention mechanism, a set of weekly attentive vectors are created. Then we applied GAT to these vectors to get the graph-based relation vector for different stocks. We could also repeat the first step on the weekly vectors and the relation vector to obtain two attentive vectors for the entire period. Lastly, we could find the relation between different market sections by applying GAT on the attentive vector.

From the above process, we could see the FinGAT applied both RNN and GAT twice. The first RNN is used for daily features collection to generate a weekly feature for each stock. The first GAT was to learn the relations between different stocks. Again, the second RNN is used to get the feature for the entire period, while the second GAT was to find the relation between different market categories. From the result in figure 4, it shows that the FinGAT performed better than other models although the LSTM has been proved as a very successful model in stock prediction.

These three mentioned models use the recurrent neural network and the graph attention network alternately. The RNN, especially GRU, is used to retrieve time series information while GAT is for getting edge weight matrix for the graph structure data. When it comes to a comparison using RNN only, an addition of GAT will giving us a better result.

## VI. FUTURE WORK AND DISCUSSION

In a rapidly changing and competitive market, the Stock prediction GAT models' success cannot be only measured by prediction accuracy, a competitive model training rate is also highly essential. By taking a step back and thinking about our research question from a practical perspective rather than a purely academic perspective, we have identified this as a key issue that needs to be addressed before the productionization of any price-prediction GNN algorithm. One such technique we could easily integrate to existing literature would be Approximate Matrix Multiplication (AMM).

### *A. Approximate Matrix Multiplication*

Multiplication of matrices - one of the most fundamental tasks not just in Graph Neural Network algorithms, but in machine learning and many other computationally expensive operations alike. There has been significant works especially in recent years directed towards efficiently approximating matrix multiplications that have enabled anywhere from 10x to 100x increase in speed when compared with the traditional sparsified, factorized, or scalar quantized methods [3], all while sacrificing only negligible amounts of accuracy. It goes without saying that exponential computational power increase

can only open new horizons for different applications of GNN's to excel in.

The paper 'Multiplying Matrices Without Multiplying'[3] presented in ICML 2021 by MIT researchers Blalock, D and Guttag, J has been one of the most recent and also the most tweeted paper in 2021 regarding to this very topic, which is the reason we have selected this paper for our review as it serves both as a summary of the recent state of AMM research, as well as offering a breakthrough improvement over the current-best performing AMM methods.

For a typical AMM task, most papers assumed the following conditions about matrices – tall, relatively dense, and residing in a single machine's memory. The main goal is usually defined as the minimization of computing resources needed to complete an approximation of linear operations with a given level of accuracy. This definition of an AMM task is due to the commonly-encountered setting where data in matrix $\boldsymbol{A}$ (holding rows of samples) need to be multiplied to matrix $\boldsymbol{B}$ (which could be an embedding matrix in the context of GNN's, or some linear operations otherwise).

The key difference which makes this paper outstanding from traditional AMM papers [4] is that traditional methods constructing the two matrices $V_A, V_B \in \mathbb{R}^{D \times d}, d \ll D$ such that $AB \approx (AV_A)(V_B^T B)$ [3]. The authors of this paper proposed the 'MADDNESS' method - a nonlinear preprocessing function which is able to reduce the problem down to a table lookup problem. With such reduction, MADDNESS no longer requires any expensive multiply-add operations, instead relying on a family of quantization functions that require zero multiply-add operations.

Nearly all existing AMM approaches rely on the projection of $\boldsymbol{A}$ and $\boldsymbol{B}$ into lower dimensional spaces and subsequently performing an exact matrix multiplication, while MADDNESS instead opted to borrow concepts from Product Quantization [11] (PQ) to approximate inner products and Euclidean distances instead.

### *B. Product Quantization*

In order to fully understand the core intuitions behind MADNESS, we should first have a decent understanding of Product Quantization (PQ) from the paper 'Product Quantization for Nearest Neighbor Search' [11].

Essentially, the idea here is to decompose a space into a Cartesian product of low dimensional subspaces to quantize each subspace separately. This then allows for easy approximations of inner products and Euclidean distances to be made, the introduction of this concept has inspired a vast majority of vector quantization methods [2] similar in principle to MADDNESS.

The intuition of PQ's inner workings is that $\boldsymbol{a^T b} \approx \boldsymbol{\hat{a}^T b}$, where $\| \boldsymbol{\hat{a} - a} \|$ results in a small value but $\boldsymbol{\hat{a}}$ is in a special structure that allows products to be calculated quickly. PQ works by preprocessing the dot products between $\boldsymbol{B}$ and 'prototype' matrices $\boldsymbol{\hat{a}}$, and reusing these precomputed values across many $\boldsymbol{A}$ vectors. When the problem is tackled in this fashion, the preprocessing function $\boldsymbol{g(A)}$ to compute the prototype $\boldsymbol{\hat{a}}$ then becomes the most computationally expensive task, this is where MADNESS differentiates itself from PQ – introducing a new $\boldsymbol{g(A)}$ function that has proven to have a large speedup across a wide range of matrix sizes.

This means that the core idea behind MADDNESS is to *determine* the most similar prototype through locality-sensitive hashing rather than a calculation in the Euclidean space. We recommend further reading about the pseudo-code and methodology used as the authors have covered it in great detail in the original paper.

The resulting takeaway from the proposed methodology is that MADDNESS performs faster and more accurate than all currently published AMM methods as shown in fig 5 and 6.

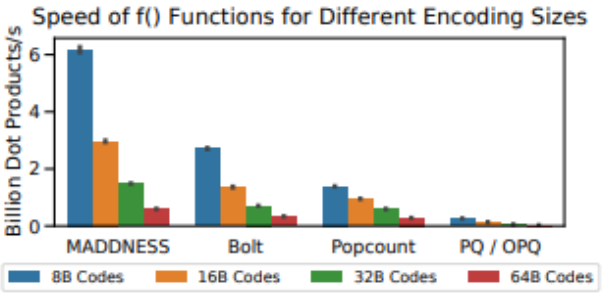


Fig. 5: MADNESS computes the approximate output twice as fast as existing methods

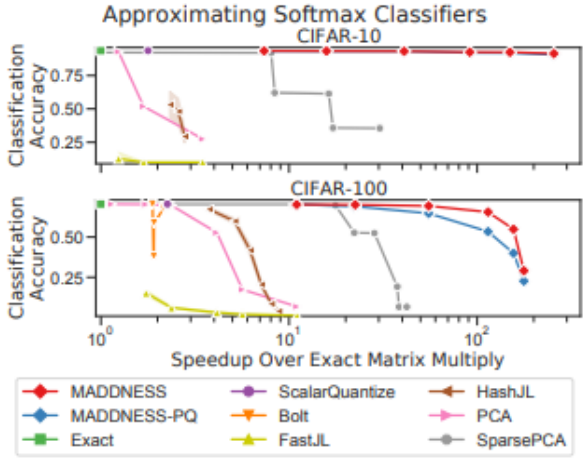


Fig. 6: MADDNESS achieves a high speed-accuracy tradeoff than any existing method when approximating two softmax classifiers

Though the results have proven to be outstanding, MADDNESS does suffer some limitations such as requiring a training set for matrix $A$, both matrices need to be tall, and that one matrix needs to be larger than the other. Additionally, the authors have acknowledged that they have not demonstrated results using convolutional layers as the weight reuse in convolutional layers presents far too '*many opportunities for algorithmic optimizations*'[3].

In summary, this paper proposed an algorithm that achieving up to 10x better performance with only a negligible amount of loss of accuracy when compared to existing methods. Seeing as the current best-performing AMM methods can offer up to 10x better performance than exact methods, this means MADDNESS can offer performance up to 100x faster than traditional exact methods within the right context. The authors also kindly included a code repository after presenting at the ICML 2021 conference, so this could be an interesting addition to try to integrate once we achieve a high enough accuracy is achieved with our model.

## VII. CONCLUSION

In this paper we have highlighted the weaknesses of traditional stock prediction ML methods, as well as a generalized overview of how graph-based models can be applied to the age-old problem of stock return prediction – we found that primitive implementations of GNN posed many weaknesses for our research problem which prompted further exploration into the more sophisticated GAT, where we then found the time-sensitivity nature of our research problem to be adequately solved by the combination of RNN and GAT as outlined in FinGAT. Furthermore we have made adequate considerations for the gaps we have identified across recent research - practicality outside of a purely academic point of view - by considering some ways to improve our model holistically beyond just a static performance indicator.

This survey of GNN literature for stock prediction has covered the relevant topics required to have a firm grasp over the research problem as well as considerations for the gaps that require further improvements beyond what has already been published. We hope this literature review has been helpful for budding academics and domain experts alike in fully understanding how the age-old problem of stock return prediction can be tackled with GNNs. Our group will be working on our own implementation inspired mostly by the FinGAT framework, and we invite the reader to join us in further contribution to this highly applicable problem.

## VIII. AUTHOR CONTRIBUTION

Kwong Chun WU contributed the Abstract, and Introduction sections, and had the additional task of combining all subsequent part together into one document, ensuring consistent formatting on all equations and diagrams.

Kam leong AO contributed all of Part III: Graph Neural Network.

Mark CHEN contributed all of Part IV: Graph Attention network.

Yong YU contributed all of part V: Time Sensitivity of GAT.

Yifei LIU contributed the Future Works & Discussion section, as well as the Conclusion, with additional efforts towards the fluidity & formatting of the final draft.

We believe all members have contributed equally, as those members who took on additional tasks to ensure the overall quality of survey were assigned less reading-intensive sections such as abstract, introduction, further work, and conclusion.

## REFERENCES


[1] Amjad, M. J. (2017). Trading Bitcoin and Online Time Series Prediction. Trading Bitcoin and Online Time Series Prediction. Published. https://devavrat.mit.edu/wp-content/uploads/2017/10/Trading-Bitcoins-and-Online-Time-Series-Prediction.pdf

[2] Andre, F., Kermarrec, A. M., & le Scouarnec, N. (2021). Quicker ADC : Unlocking the Hidden Potential of Product Quantization With SIMD. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(5), 1666–1677. https://doi.org/10.1109/tpami.2019.2952606

[3] Blalock, D. (2021). Multiplying Matrices Without Multiplying. 38th International Conference on Machine Learning. Published. https://arxiv.org/pdf/2106.10860.pdf

[4] Blalock, D. W., & Guttag, J. V. (2017). Bolt. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Published. https://doi.org/10.1145/3097983.3098195

[5] Brody, S. (2021). How Attentive are Graph Attention Networks? How Attentive Are Graph Attention Networks? Published.

[6] Ciaian, P. (2014). The Economics of BitCoin Price Formation. The Economics of BitCoin Price Formation. Published. https://arxiv.org/ftp/arxiv/papers/1405/1405.4498.pdf

[7] Cirstea, R. (2021). Graph Attention Recurrent Neural Networks for Correlated Time Series Forecasting—Full Version. Graph Attention Recurrent Neural Networks for Correlated Time Series Forecasting—Full Version. Published. https://arxiv.org/pdf/2103.10760.pdf

[8] Estimate of overall cryptocurrency market cap per week from July 2010 to June 2021. (2021). Statista. https://www.statista.com/statistics/730876/cryptocurrency-maket-value/

[9] Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., & Yin, D. (2019). Graph Neural Networks for Social Recommendation. The World Wide Web Conference on - WWW '19. Published. https://doi.org/10.1145/3308558.3313488

[10] Hsu, Y. (2021). FinGAT: Financial Graph Attention Networks for Recommending Top-K Profitable Stocks. FinGAT: Financial Graph Attention Networks for Recommending Top-K Profitable Stocks. Published. https://arxiv.org/abs/2106.10159

[11] Jégou, H., Douze, M., & Schmid, C. (2011). Product Quantization for Nearest Neighbor Search. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(1), 117–128. https://doi.org/10.1109/tpami.2010.57

[12] Jin, W., Li, Y., Xu, H., Wang, Y., Ji, S., Aggarwal, C., & Tang, J. (2021). Adversarial Attacks and Defenses on Graphs. ACM SIGKDD Explorations Newsletter, 22(2), 19–34. https://doi.org/10.1145/3447556.3447566

[13] Kristoufek, L. (2013). BitCoin meets Google Trends and Wikipedia: Quantifying the relationship between phenomena of the Internet era. Scientific Reports. Published. https://www.nature.com/articles/srep03415#Sec8

[14] Ma, Y., Guo, Z., Ren, Z., Tang, J., & Yin, D. (2020). Streaming Graph Neural Networks. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. Published. https://doi.org/10.1145/3397271.3401092

[15] Mallqui, D. C., & Fernandes, R. A. (2019). Predicting the direction, maximum, minimum and closing prices of daily Bitcoin exchange rate using machine learning techniques. Applied Soft Computing, 75, 596–606. https://doi.org/10.1016/j.asoc.2018.11.038

[16] Patil, P., Wu, C. S. M., Potika, K., & Orang, M. (2020). Stock Market Prediction Using Ensemble of Graph Theory, Machine Learning and Deep Learning Models. Proceedings of the 3rd International Conference on Software Engineering and Information Management. Published. https://doi.org/10.1145/3378936.3378972

[17] Polasik, M. (2015). Price Fluctuations and the Use of Bitcoin: An Empirical Inquiry. International Journal of Electronic Commerce. Published. https://doi.org/10.1080/10864415.2016.1061413

[18] Vaswani, A. (2017). Attention is all you need. 31st Conference on Neural Information Processing Systems. Published.

[19] Velickovic, P. (2018). GRAPH ATTENTION NETWORKS. ICLR. Published. https://arxiv.org/pdf/1710.10903.pdf

[20] Zhang, X. (2020). Traffic Flow Forecasting with Spatial-Temporal Graph Diffusion Network. AAAI Conference on Artificial Intelligence. Published. https://ojs.aaai.org/index.php/AAAI/article/view/17761

[21] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. AI Open, 1, 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001