Integrating Graph and Recurrent Neural Networks for Spatiotemporal Reasoning

Victoria Magdalena Dax[1], Zhi Li[2], Xiaowei Zhang[2], Hemabh Shekhar[2], Jiachen Li[3], and Mykel J. Kochenderfer[1]

***Abstract*— Combining graph neural networks (GNNs) with recurrent neural networks (RNNs) offers a promising approach to enhance spatiotemporal reasoning in diverse applications, from autonomous driving to medical assistive devices. However, the best way to combine these networks is not straightforward. This paper introduces two new approaches to combining GNNs and RNNs in ways that differ from traditional sequential arrangements. We explore embedding one network within the other, resulting in a more integrated representation of spatial and temporal features. We show that these merged architectures outperform traditional models on multiple datasets. The R-GNN architecture, which integrates a GNN message-passing layer into the new memory content formation of a GRU cell, in particular, performs best, highlighting the potential of architectural fusions to improve spatiotemporal reasoning.**

I. Introduction

Graph neural networks (GNNs) and recurrent neural networks (RNNs) have each demonstrated their effectiveness in representing graph-structured and sequential data, respectively. Integrating these two models offers the potential to enhance spatio-temporal reasoning with applications ranging from automated driving to healthcare monitoring. However, combining these networks is a complex task. Previous attempts have primarily followed sequential arrangements, i.e., starting with an RNN block followed by a GNN block or vice versa. Although these methods have yielded promising results, they fail to fully exploit the synergistic potential of GNNs and RNNs.

Spatiotemporal graph modeling has been widely studied to capture the dynamic relations between interacting agents. Social-STGCNN [1] introduced a spatio-temporal graph convolutional neural network for human trajectory prediction. EvolveGraph [2] and dNRI [3] are dynamic relational reasoning methods for identifying underlying relations based on the information encoded in given trajectory sets. A spatiotemporal attention mechanism [4] and a graph transformer network [5] were proposed to model the intergraph temporal dependencies.

Combining GNNs and RNNs has found successful application in various domains such as traffic prediction [6], [7], fault diagnostics [8], and video segmentation [9]. Recent work has explored the effectiveness of these sequential architectures: [10] is an end-to-end interaction modeling framework with a focus on interpretability, while GCRN [11] extended classical RNNs to graph-structured data, showing promise in language tasks. EvolveGraph [2] offered a trajectory forecasting model that emphasizes prediction of relational structures among interactive agents. STSGCN [12] defines independent RNN modules for different time periods to capture the heterogeneities of localized spatiotemporal graphs.

These previous approaches primarily rely on a layered sequence of GNN and RNN blocks, potentially limiting the depth of integration between spatial and temporal features. We address this limitation by proposing alternative architectures that embed one network within the other. This paper proposes two novel layer architectures that integrate GNNs and RNNs in a way that can lead to more robust spatiotemporal reasoning capabilities. Our main contributions are:

1) We introduce two novel layer architectures that integrate GNNs and RNNs.
2) We demonstrate enhanced spatiotemporal reasoning capabilities through these integrated architectures.
3) We analyze our approach across applications in diverse domains spanning transport, epidemiology, and robotics.

We address the first two contributions in Section III and delve into the empirical validation in Section IV.

This work focuses on layer-level reasoning. While many recent works [13]–[15] explore the combination of RNNs and GNNs, they do so by stacking existing layers of either type. Our objective is *not* to compete with current state-of-the-art models but rather to introduce layer architectures that deeply integrate RNNs and GNNs. This approach enables more efficient and accurate processing of dynamic graphs.

II. Preliminaries

Graph neural networks (GNNs) are a class of deep learning models designed to process graph-structured data. GNNs process graph-structured data by iteratively passing and aggregating messages between neighboring nodes within a graph $\mathcal{G}(V, E)$. Here, V is the set of n nodes and E is the set of m edges. Each node $u \in V$ has input features $x_u \in \mathbb{R}^d$, which is the uth row of feature matrix X, and a label $y_u \in \mathbb{R}$, which is a function of $\{x_u\} \cup \{x_v \mid v \in \mathcal{N}_u\}$ where $\mathcal{N}_u$ is the set of neighboring nodes of u. The message-passing update

$$x_u^{(k+1)} = \textsc{up}^{(k)}\left(x_u^{(k)}, \textsc{agg}^{(k)}\left(\{x_v^{(k)} \mid v \in \mathcal{N}_u\}\right)\right), \quad (1)$$

*This work was conducted in part during an internship at Uber Inc., and we gratefully acknowledge their support and resources, which were integral to the completion of this research.

[1] V. M. Dax and M. J. Kochenderfer are with the Stanford Intelligent Systems Laboratory (SISL), Stanford University, USA {vmdax, mykel}@stanford.edu

[2] Z. Li, X. Zhang, and H. Shekhar are with Uber Inc., USA. {lizhi, xiaowei.zhang, hemabh}@uber.com

[3] J. Li is with the University of California, Riverside, USA. jiachen.li@ucr.edu

where UP (short for *update*) and AGG (short for *aggregate*) denote arbitrary differentiable functions (e.g., MLPs), and k denotes the index into the message-passing layers, can be applied iteratively to capture information from a broader neighborhood. The final node representations can then be used for various tasks, such as node classification and link prediction, or, when aggregated, for graph classification. Kipf and Welling [16] introduced a more scalable approach based on an efficient variant of graph convolution called the *graph convolutional network* (GCN) layer:

$$h_{\text{GCN}}(X; W) = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}XW \tag{2}$$

where $\hat{A} = A + \mathbb{I}$ is the adjacency matrix with inserted self-loops, $\hat{D}$ is its diagonal degree matrix, and W is the learned weight matrix.

Recurrent neural networks (RNNs) are designed for sequential data analysis, such as time series or natural language. They maintain an internal state that captures previous inputs, making them suitable for sequence processing. A common drawback with traditional RNNs is the vanishing gradient problem [17], which limits recognizing long-range dependencies within sequences.

Gated recurrent units (GRUs) [18] use gating mechanisms to regulate information flow, facilitating prolonged memory retention and improved sequence training. GRUs have four components: the reset and update gates, followed by the new memory content gate, and the final hidden state. The *reset* r_t and *update gates* z_t determine how much of the previous memory to keep or discard and how much of the hidden state to update with the new one, respectively. Both gates consist of a linear layer with a sigmoid activation that scales the output into a [0,1] interval, e.g.,

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r). \tag{3}$$

Here, h_{t-1} is the hidden state from the previous time step and x_t is the input at the current time step. W_r and b_r are the learned weight matrix and bias vector, respectively. The *new memory content* $\tilde{h}_t$, which combines the input and the previous hidden state modified by the reset gate, is

$$\tilde{h}_t = \sigma(W \cdot [r_t \odot h_{t-1}, x_t] + b), \tag{4}$$

where $r_t \odot h_{t-1}$ is the element-wise multiplication of the reset gate and the previous hidden state. This allows the model to forget parts of the previously memorized hidden state. Finally, the *final hidden state* h_t combines the previous hidden state and the new memory content

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{5}$$

based on the update gate z_t, which determines the proportion of old memory h_{t-1} to retain versus the new memory $\tilde{h}_t$.

III. Method

The integration of GNNs and RNNs has been approached by either processing spatial information before temporal (GNN-RNN), or, the inverse, i.e., processing temporal information before spatial (RNN-GNN). Our research posits that a thighter integration of these architectures can lead to better representations. Instead of a sequence, we suggest embeddeding one network within the other.

Algorithm 1 G-RNN Propagate

Input: X, E, h_0
$X, h_0 \leftarrow \text{GRU}([X, h_0], E)$
$E \leftarrow \text{addSelfLoops}(E)$
$\hat{E} \leftarrow \text{normalize}(E)$
$Y \leftarrow \text{update}(X, \hat{E})$
return Y, h_0

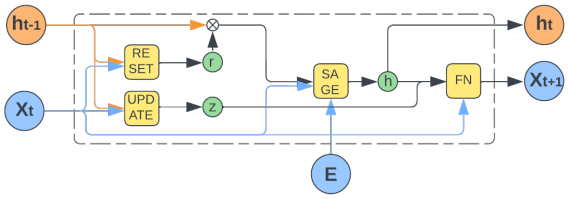

Fig. 1. G-RNN internal gating mechanism.

Embedding an RNN within a GNN (G-RNN) enhances temporal expressivity within each neighborhood aggregation step. Each message incorporates a learned temporal summary, allowing the GNN to pass time-aware information through the graph. Conversely, embedding a GNN within a GRU (R-GNN) allows graph-structured context to directly modulate memory updates. The candidate hidden state becomes a graph-informed summary of the current input, increasing the capacity to learn spatially coherent trajectories.

We stipulate that these embedded architectures increase the function class capacity by allowing for localized recurrence (in G-RNN) or structured nonlinearity in gating (in R-GNN), akin to adding depth or hierarchy. While a full expressivity or convergence analysis is out of scope, our empirical results support the hypothesis that such cross-modal embedding improves modeling of spatiotemporal dependencies beyond sequential baselines.

A. RNN Embedded within a GNN (G-RNN).

Analyzing the GCN layer from Equation (2), the update function can be disassembled into three core components: a message function, an aggregation function, and an activation function. The message $W^{(l)}h_u^{(l)}$ is a linear transformation that is applied to each node feature. The aggregation function is (usually) a summation, and the chosen activation will vary based on the problem space and task.

Conventionally, the feature matrix $H^{(l)}$ has dimensions $N \times d$, where N is the number of nodes in the graph and d is the dimensionality of each node's feature vector. This study proposes an extension of this framework by incorporating temporal variations. Instead of focusing on static features, we accommodate trajectories. This modification results in $H^{(l)}$ being a three-dimensional tensor of size $N \times T \times d$, where T denotes the number of time steps. Our approach uses an RNN

in place of the linear layer in the message update function

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} GRU(h_u^{(l+1)})\right). \quad (6)$$

Such an integration requires an input with an associated hidden state. This approach is outlined in Algorithm 1 and visualized in Section III-A.

B. GNN Embedded within a RNN (R-GNN).

An alternative approach involves embedding a GNN block into the gating mechanisms of a traditional RNN, specifically, a GRU. GRUs have fewer parameters than LSTMs [19], allowing them to train faster with less data. In practice, GRUs have been shown to perform comparably to LSTMs on various tasks [20], with the choice often depending on the specific application and data.

Our method, which we call *memory content embedding*, integrates a GCN within the new memory content $\tilde{h}_t$ gate. We recall that the reset gate decides the proportion of past memory to retain, while the update gate determines how the prior hidden state gets updated with a newly proposed state. The new memory content $\tilde{h}_t$, which is now being processed by a GNN, encodes the current information x_t and the reset hidden state. The final hidden state h_t then integrates old and new memory based on the update gate. This information flow is outlined in Algorithm 2, where $\odot$ represents elementwise multiplication. Essentially, we replaced the new memory content evaluation from Equation (4) with a GNN:

$$\tilde{h}_t = GNN([r_t \odot h_{t-1}, x_t]) \quad (7)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (8)$$

Instead of using a linear layer paired with a $\tanh$ activation, we introduce a GCN to handle the spatial dynamics within the temporal gating mechanism of the RNN. And, Section III-B visualizes the information flow of this new layer type.

Changing the input transformation, i.e., the reset gate of Equation (3), by introducing a GCN at that level would be equivalent to a *sequential embedding*. While this would essentially be a single-layer version of the architecture used by Graber and Schwing [3], stacking this *sequential* layer would result in an alternating GNN-RNN pattern that has not been explored.

Algorithm 2 R-GNN Propagate

Input: X, E, h_0
$r \leftarrow \text{reset}(X, h_0)$
$z \leftarrow \text{update}(X, h_0)$
$E \leftarrow \text{addSelfLoops}(E)$
$n \leftarrow \text{GCN}([X \| r \odot h_0], E)$
return $(1 - z) \odot n + z \odot h_0$

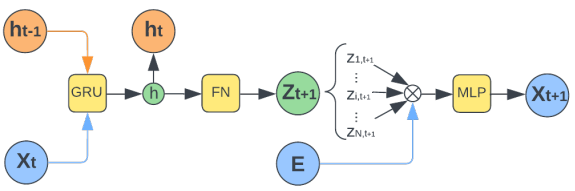

Fig. 2. R-GNN message-passing mechanism.

IV. Experiments

To assess the effectiveness of our method, we conduct a series of experiments [1] involving regression and classification. We compare our method against six baseline methods.

1) Two naive approaches: A *rolling average* from the most recent N time steps and a *zero baseline*, which predicts no change from the last observed data point.
2) A *non-linear predictor*, a two-layer perceptron (MLP) with ReLU activation, which is easy to train and performs decently when predicting increments, i.e., predicting dX_{t+1} instead of X_{t+1}.
3) A two-layer *LSTM* with ReLU activation. This is a simple architecture, which isolates the relevance of the temporal dimension of the data.
4) Both traditional approaches of sequentially stacking GNNs and RNNs. First, a GCN block extracts spatial features from the graph structure, then an LSTM module processes the data's temporal aspects. We also evaluate the reverse arrangement, i.e., a LSTM followed by a GCN. Through concatenation, these two architectures can interpret both spatial relations and temporal correlations within the data.

As in prior work [2], [21], we evaluate the standard metrics used in trajectory forecasting: 1) the average displacement error (ADE), which refers to the mean Euclidean distance between the ground truth and predicted trajectories, 2) the final displacement error (FDE), which refers to the Euclidean distance between the predicted final position and the ground truth at the prediction horizon, 3) the root mean squared error (RMSE), and 4) the mean absolute error (MAE).

Because our dataset has many extended zero-sequences, we also report accuracy, precision, and recall. These are based on whether we predict activation. Otherwise, the zero baseline might be considered a promising, no-effort solution. We address the significant data imbalance with metrics that capture missed deficits. We evaluate *masked*-MAE and *masked*-MSE.

A. Supply-Deficit Prediction

The gig economy, particularly ride-sharing services, often faces challenges in matching driver supply with passenger demand. The concept of an "undersupplied" supply hour quantifies the additional number of drivers needed at a specific time and location to meet demand. "Open supply hours" are determined by tallying the minutes a driver spends available (en route or idle) in a designated area. Therefore, a supply deficit signifies a shortage of drivers necessary to fulfill demand. For example, a deficit of 1.5 indicates the need for 1.5 additional drivers during that period.

[1]The code is publicly available at `https://github.com/victorialena/gRNN`

TABLE I

[SUPPLY DEFICIT] PERFORMANCE COMPARISON FOR DIFFERENT LAYER TYPES

Metrics	MAE [m]	RMSE [m]	ADE [m]	FDE [m]	Train Loss [m^2]
Rolling Avg	0.1275	0.2276	0.1275	0.1254	–
Zero Baseline	**0.0524**	0.1791	0.0524	0.0513	–
MLP	0.0666	0.1496	0.0666	0.0613	0.0273
LSTM	0.0592	0.1465	0.0592	0.0585	0.0269
GNN2RNN	0.0586	**0.1452**	0.0586	**0.0550**	0.0237
RNN2GNN	0.0580	0.1464	0.0580	0.0563	0.0252
GRNN	0.0590	0.1533	0.0590	0.0581	0.0290
RGNN	**0.0553**	0.1456	**0.0553**	**0.0550**	0.0250
	Accuracy	Precision	Recall	mMAE [m]	mMSE [m^2]
Rolling Avg	0.5714	0.2538	0.8810	0.1295	0.2613
Zero Baseline	**0.8425**	0.0000	0.0000	0.2056	0.3280
MLP	0.3714	0.1976	**0.9669**	0.1167	**0.2304**
LSTM	0.5804	0.2660	0.9162	0.1183	0.2325
GNN2RNN	0.6207	0.2786	0.8795	0.1111	0.2254
RNN2GNN	0.6766	0.3126	0.8474	0.1148	0.2276
GRNN	0.7638	**0.3713**	0.7264	0.1248	0.2419
RGNN	0.7235	0.3407	0.7972	**0.1146**	**0.2306**

By implementing a hexagonal indexing system, geographical areas can be divided into a grid of hexagons, or "hexes," of uniform size. This hexagonal layout offers a significant advantage over traditional square grids because it more closely approximates a circular shape, both for individual cells and clusters of adjacent cells, preserving the concept of radius. A "hex cluster" refers to a collection of neighboring hexes. For context, a city such as San Francisco is composed of approximately 400 clusters, with each cluster containing tens of hexes. Clusters are interconnected with undirected edges if they share a common boundary. Each node is characterized by a 32-dimensional feature vector, including time of day (encoded in sine-cosine form), the day of the week (one-hot encoded), the number of encompassed hexes, supply deficit, surplus, active drivers, and users in search of a ride.

The dataset contains six weeks of data, translating to approximately 1M data points after hourly aggregation. We use a sliding window approach, examining patterns over 50 consecutive hours to predict the following 25 steps, with a 4-hour offset between each window. In this approach, the analysis window shifts by a set number of steps (4 hours in our case) to create overlapping segments of data for training the model. To prevent data leakage and ensure the integrity of our analysis, the data is split into training and testing sets before applying the sliding window method.

Table I shows that graph-based methods, most notably GNN2RNN and RGNN, outperform all other methods by 6-24%, depending on the metric. Their superior performance underscores their potential advantages over traditional methods such as MLP and LSTM. However, when comparing RGNN, our newly proposed architecture, with GNN2RNN, the more traditional sequential arrangement, we only lose about 1% performance in some metrics while gaining 6-8% on average in others. Interestingly, the zero-baseline takes achieves the lowest MAE. Since MAE is less sensitive to outliers than MSE, the baseline can simply exploit the sparsity of the data.

It is important to balance precision and recall. While the MLP baseline achieves the highest recall – indicating strong coverage of positive instances – its low precision suggests a high rate of false positives. Configurations such as GNN2RNN and RGNN better balance precision and recall, making them potentially more suitable for applications where both false positives and false negatives carry significant consequences.

The relatively lower training losses in configurations like GNN2RNN and RGNN suggest room for potential enhancement through additional training or larger datasets. While the top-performing models have been bolded for clarity, models within 5% of the top scores can offer predictive value.

B. COVID-19 Open Dataset

The COVID-19 Open Dataset (COD)[2] offers an extensive resource for researchers and analysts studying the pandemic's global impact. It consolidates epidemiological data from 22,579 unique locations across 232 countries and territories, including state and county-level data. This dataset covers hospitalizations and vaccinations, as well as mobility trends and non-pharmaceutical interventions, such as lockdowns. We also added population statistics[3], and the 2023 Commonwealth Fund health raking report[4]. We are interested in analyzing how different states have handled the COVID-19 health crisis.

[2]https://health.google.com/covid-19/open-data

[3]https://www.census.gov/quickfacts

[4]https://www.commonwealthfund.org/publications/scorecards

TABLE II
[COVID-19] PERFORMANCE COMPARISON FOR DIFFERENT LAYER TYPES

Metrics	MAE [cm]	RMSE [cm]	ADE [cm]	FDE [cm]	Train Loss [cm^2]
Rolling Avg	0.498 ± 0.091	1.580 ± 0.269	0.825 ± 0.148	0.818 ± 0.152	–
ZeroBaseline	0.802 ± 0.136	2.357 ± 0.328	1.273 ± 0.213	1.256 ± 0.293	–
MLP	0.809 ± 0.132	2.419 ± 0.280	1.287 ± 0.207	1.290 ± 0.291	6.035 ± 0.181
LSTM	0.784 ± 0.072	1.787 ± 0.229	1.258 ± 0.114	1.243 ± 0.168	3.066 ± 0.132
RNN2GNN	0.886 ± 0.113	1.884 ± 0.287	1.421 ± 0.184	1.344 ± 0.134	3.171 ± 0.307
GNN2RNN	1.026 ± 0.214	2.612 ± 1.513	1.685 ± 0.410	1.915 ± 1.151	8.367 ± 8.445
GRNN	0.751 ± 0.131	2.103 ± 0.348	1.216 ± 0.208	1.197 ± 0.239	4.703 ± 0.461
RGNN	**0.667 ± 0.102**	**1.774 ± 0.227**	**1.076 ± 0.167**	**1.046 ± 0.190**	2.960 ± 0.140
	Accuracy	Precision	Recall	mMSE [cm^2]	mADE [cm]
Rolling Avg	0.787 ± 0.021	0.774 ± 0.020	0.995 ± 0.005	3.141 ± 1.116	0.457 ± 0.075
Zero Baseline	0.282 ± 0.039	0.000 ± 0.000	0.000 ± 0.000	7.829 ± 2.033	0.886 ± 0.131
MLP	0.283 ± 0.039	0.811 ± 0.134	0.002 ± 0.001	8.036 ± 1.850	0.893 ± 0.127
LSTM	**0.654 ± 0.021**	0.741 ± 0.033	**0.796 ± 0.032**	**3.776 ± 1.052**	0.686 ± 0.058
RNN2GNN	0.634 ± 0.013	0.749 ± 0.028	0.736 ± 0.020	3.981 ± 1.045	0.806 ± 0.063
GNN2RNN	0.651 ± 0.032	0.746 ± 0.037	0.777 ± 0.041	3.462 ± 1.077	0.693 ± 0.074
GRNN	0.384 ± 0.054	0.783 ± 0.023	0.183 ± 0.079	6.112 ± 2.036	0.796 ± 0.125
RGNN	0.356 ± 0.038	**0.834 ± 0.037**	0.125 ± 0.041	4.049 ± 1.000	**0.689 ± 0.089**

Therefore, the graph has 51 nodes (i.e., one for each state plus the District of Columbia), and two nodes are connected if the respective states share a border.

Table II summarizes the results. The rolling average is performing best across multiple metrics, suggesting that, contrary to Section IV-A, a machine learning approach might not be necessary for this specific problem domain. RGNN is again outperforming all other baselines, as well as G-RNN. It outperforms the LSTM by 21% in MAE and 25–30% in ADE and FDE. Further, we note that RGNN has the highest precision at 0.9, which comes at the cost of a precision-recall trade-off. The LSTM model has a balanced precision-recall trade-off with around 0.7 for both values. This balanced binary prediction closes the performance gap in mMSE and mADE. The good performance of LSTM suggests that state policies, actions, and demographics are not affecting the performance of neighboring states in mitigating outbreaks.

C. Motion Dataset

We evaluate our proposed method using motion capture recordings from the CMU Motion Capture Database[5]. This database provides high-resolution recordings from a system with 12 infrared cameras operating at 120 Hz, capturing detailed movements across 31 distinct joints of the subject's body. For each test subject, the database includes both bone length and root position data. Additionally, the motion data is categorized by the performed action. We selected the recordings of a variety of test subjects walking, running, jumping, dancing, and playing basketball. It is important to note, however, that the dataset exhibits a notable imbalance: approximately 25% of the data is related to walking, which may impact the generalizability of the findings. Since this is not a sparse dataset, we do not report classification and masked metrics. However, because it has a multimodal prediction dimension, we report both MAE and MSE.

The RGNN model performs best across multiple metrics in Table III. This model has the lowest mean absolute error (MAE) at 0.5382, underscoring its capability to minimize average prediction errors more effectively compared to the trailing RNN2GNN model. Moreover, RGNN improves 8–15% in MSE and RMSE. These metrics highlight RGNN's consistency not only in delivering accurate average predictions but also in its precision with individual forecasts, especially for outliers or larger error predictions, where MSE's sensitivity to such errors is crucial. RGNN's superiority extends to ADE and FDE, where it beats the next best baseline by 7%.

Across all datasets, our evaluation demonstrates that integrating GNNs within a GRU framework a) outperforms its inverse (G-RNN) and b) improves upon traditional approaches such as the RNN2GNN and LSTM architectures, highlighting the advantage of embedding relational reasoning within recurrent architectures. In scenarios where a more conventional sequential model is used, prioritizing temporal processing ahead of relational reasoning appears to be preferable.

V. CONCLUSION

This paper examines the structural integration of GNNs and RNNs to improve performance and efficiency on spatiotemporal reasoning tasks. Prior approaches have combined these architectures sequentially, treating spatial and temporal modeling as separate stages. Our findings suggest that a deeper integration, beyond simple sequencing, can improve performance with graph-based time-series forecasting.

[5]http://mocap.cs.cmu.edu

TABLE III

[MOTION] PERFORMANCE COMPARISON FOR DIFFERENT LAYER TYPES

Metrics	MAE [m]	MSE [m^2]	RMSE [m]	ADE [m]	FDE [m]
Constant Avg	3.605 ± 0.448	53.568 ± 12.798	7.276 ± 0.851	8.910 ± 1.105	10.751 ± 1.355
Rolling Avg	1.581 ± 0.150	10.424 ± 2.097	3.215 ± 0.315	3.654 ± 0.350	5.714 ± 0.594
MLP	0.818 ± 0.082	3.052 ± 0.650	1.738 ± 0.186	1.772 ± 0.183	3.080 ± 0.338
LSTM	0.659 ± 0.072	2.066 ± 0.466	1.430 ± 0.159	1.364 ± 0.134	2.813 ± 0.271
RNN2GNN	0.661 ± 0.055	2.098 ± 0.727	1.430 ± 0.245	1.365 ± 0.118	2.827 ± 0.212
GNN2RNN	0.923 ± 0.081	3.018 ± 1.194	1.709 ± 0.334	1.899 ± 0.165	2.954 ± 0.283
GRNN	0.654 ± 0.061	**1.828 ± 0.534**	**1.340 ± 0.194**	1.345 ± 0.127	2.701 ± 0.234
RGNN	**0.619 ± 0.068**	2.353 ± 1.372	1.480 ± 0.433	**1.279 ± 0.143**	**2.694 ± 0.284**

We introduced two architectures in which one network type is embedded within the other. Our experimental results show that our integrated models generally outperform traditional methods. RGNN, in particular, achieves the lowest MAE (by 12–25%), ADE (by 8–18%), and FDE (by 7–16%), as well as a more balanced precision-recall trade-off, indicating superior predictive accuracy and reliability.

We have focused on dynamic graphs, more specifically, graphs with temporally evolving node features. However, an important limitation of our study is that we did not consider scenarios where the graph structure itself evolves, i.e., where nodes and edges can appear and disappear over time. Research such as Fully Dynamic Graph Neural Network (FDGNN) [22] and EvolveGCN [23] specifically address these types of fully dynamic graphs.

Given the applicability of these architectures across diverse applications, this research underscores the necessity for continued exploration and refinement of integrated neural architectures. Also, formal characterization of representational power and convergence properties of embedded GNN-RNN architectures remains an open direction.

REFERENCES

[1] A. Mohamed, K. Qian, M. Elhoseiny, and C. Claudel, "Social-STGCNN: A social spatio-temporal Graph Convolutional Neural Network for Human Trajectory Prediction," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[2] J. Li, F. Yang, M. Tomizuka, and C. Choi, "EvolveGraph: Multi-Agent Trajectory Prediction with Dynamic Relational Reasoning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] C. Graber and A. Schwing, "Dynamic Neural Relational Inference," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[4] J. Li, H. Ma, Z. Zhang, J. Li, and M. Tomizuka, "Spatio-temporal graph dual-attention network for multi-agent prediction and tracking," in *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[5] C. Yu, X. Ma, J. Ren, H. Zhao, and S. Yi, "Spatio-temporal graph transformer networks for pedestrian trajectory prediction," in *European Conference on Computer Vision (ECCV)*, 2020.

[6] C. Chen, K. Li, S. G. Teo, *et al.*, "Gated Residual Recurrent Graph Neural Networks for Traffic Prediction," in *AAAI Conference on Artificial Intelligence*, 2019.

[7] L. Zhao, Y. Song, C. Zhang, Y. Liu, *et al.*, "T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, pp. 3848–3858, 2018.

[8] B. L. H. Nguyen, T. V. Vu, T.-T. Nguyen, M. Panwar, and R. Hovsapian, "Spatial-Temporal Recurrent Graph Neural Networks for Fault Diagnostics in Power Distribution Systems," *IEEE Access*, vol. 11, pp. 46 039–46 050, 2022.

[9] E. Brissman, J. Johnander, M. Danelljan, and M. Felsberg, "Recurrent Graph Neural Networks for Video Instance Segmentation," *International Journal of Computer Vision*, vol. 131, pp. 471–495, 2022.

[10] E. Sachdeva and C. Choi, "DIDER: Discovering Interpretable Dynamically Evolving Relations," *IEEE Robotics and Automation Letters*, vol. 7, pp. 1–8, Oct. 2022.

[11] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured Sequence Modeling with Graph Convolutional Recurrent Networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[12] C. Song, Y. Lin, S. Guo, and H. Wan, "Spatial-Temporal Synchronous Graph Convolutional Networks: A New Framework for Spatial-Temporal Network Data Forecasting," in *AAAI Conference on Artificial Intelligence*, 2020.

[13] J. Li, C. Hua, J. Park, H. Ma, V. Dax, and M. J. Kochenderfer, "Evolvehypergraph: Group-aware dynamic relational reasoning for trajectory prediction," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[14] D. Montero and J. Yebes, "Combining graph and recurrent networks for efficient and effective segment tagging," in *Proceedings of the Learning on Graphs Conference*, 2022.

[15] N. S. Roudbari, Z. Patterson, U. Eicker, and C. Poullis, "Simpler is better: Multilevel abstraction with graph convolutional recurrent neural network cells for traffic prediction," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2022.

[16] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[17] R. Pascanu, T. Mikolov, and Y. Bengio, "On the Difficulty of Training Recurrent Neural Networks," in *International Conference on Machine Learning (ICML)*, 2013.

[18] K. Cho, B. Van Merriënboer, C. Gulcehre, *et al.*, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–80, 1997.

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint*, 2014.

[21] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural Relational Inference for Interacting Systems," in *International Conference on Machine Learning*, 2018.

[22] A. Moallemy-Oureh, S. Beddar-Wiesing, R. Nather, and J. M. Thomas, "FDGNN: Fully dynamic graph neural network," *arXiv preprint*, 2022.

[23] A. Pareja, G. Domeniconi, J. Chen, T. Ma, *et al.*, "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.