GRAPH SIGNAL GENERATIVE DIFFUSION MODELS

Yiğit Berkay Uslu[†] Samar Hadou[†] Sergio Rozada^{*} Shirin Saeedi Bidokhti[†] Alejandro Ribeiro[†]

[†]Dept. of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, USA *Dept. of Signal Processing and Communications, King Juan Carlos University, Madrid, Spain

ABSTRACT

We introduce U-shaped encoder-decoder graph neural networks (U-GNNs) for stochastic graph signal generation using denoising diffusion processes. The architecture learns node features at different resolutions with skip connections between the encoder and decoder paths, analogous to the convolutional U-Net for image generation. The U-GNN is prominent for a pooling operation that leverages zero-padding and avoids arbitrary graph coarsening, with graph convolutions layered on top to capture local dependencies. This technique permits learning feature embeddings for sampled nodes at deeper levels of the architecture that remain convolutional with respect to the original graph. Applied to stock price prediction—where deterministic forecasts struggle to capture uncertainties and tail events that are paramount—we demonstrate the effectiveness of the diffusion model in probabilistic forecasting of stock prices.

Index Terms— Diffusion models, stochastic graph signals, graph neural networks, stock price prediction

1. INTRODUCTION

Data defined on irregular domains is pervasive, with applications ranging from recommender systems to wireless networks [1, 2, 3, 4]. This paper addresses the problem of generating signals (data) on a given graph when the underlying distribution is unknown and may not admit a tractable form. To that end, we leverage generative diffusion models [5, 6], where the central idea is to gradually corrupt a sample from the original distribution with noise in a forward process until it becomes indistinguishable from a simple, non-informative distribution, e.g. Gaussian. By reversing this process, the noise can be transformed back into samples of the original distribution. The backward dynamics rely on a parametric denoiser that removes noise at different noise levels to progressively recover the clean signal [5, 6]. While this approach has achieved remarkable success in image [7, 8] and graph generation [9, 10], it has received less attention in the setting where signals are defined on a fixed, given graph.

Graph signal generation has been explored across diverse domains such as recommender systems [11, 12, 13], spatio-temporal forecasting [14, 15, 16], finance [17, 18], or wireless communications [19, 20, 21]. In a nutshell, these works adapt the successful denoising diffusion probabilistic model (DDPM) framework [5] from computer vision to graphs, parametrizing the backward process with graph neural networks (GNNs) [22, 23], given that signals are defined on a known graph. Despite these advances, existing approaches are application-driven, crafting tailored solutions that are usually used to solve specific downstream tasks. As a result, most

methods parametrize the backward process with architectures that blend domain-specific knowledge with standard GNNs [12, 14, 16] or graph attention mechanisms [15, 17]. However, architecture design should transcend application domains and be guided by the requirements of diffusion. Evidence from image generation highlights the importance of this choice, with U-Net [24] emerging as the canonical design by combining encoder pooling for global context with decoder upsampling and skip connections for fine detail [5]. A comparable architectural design for graph-based domains is required to establish a general framework for graph-signal diffusion beyond application-specific solutions.

Motivated by this landscape, we propose a general generative modeling framework that extends DDPM for graph signals and unifies the main approaches across application domains. Furthermore, we introduce a novel architecture that generalizes the U-Net architecture to graph convolutions by introducing a pooling scheme that leverages zero-padding and nested sampling matrices to construct valid graph convolutional features and aggregation neighborhoods supported over the original graph. To illustrate the framework, we consider stock price prediction, a setting where generative models can capture uncertainties and tail events that deterministic forecasts overlook, and where capturing uncertainties is valuable not only for prediction but also for risk management [25]. The summary of the contributions is as follows.

- C1. We formulate a generative diffusion model for stochastic graph signals that unifies existing application-driven approaches
- C2. We propose a novel GNN architecture, termed U-GNN, that leverages graph convolutions and zero-padding graph pooling to capture local dependencies at different resolutions.
- C3. We validate our method on stock-price forecasting, where stochastic predictions are vital to represent uncertainties.

2. DIFFUSION MODELS FOR GRAPH SIGNALS

Consider a weighted, undirected graph $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathcal{W})$, where \mathcal{V} is the node set with $|\mathcal{V}|=N,\mathcal{E}$ is the edge set, and $\mathcal{W}:\mathcal{E}\mapsto\mathbb{R}$ is a function assigning weights to edges. We represent graph-structured data with a graph signal $\mathbf{X}\in\mathbb{R}^{N\times F}$, where F is the number of features per node. Equivalently, we use a vectorized form $\mathbf{x}\in\mathbb{R}^{NF}$, obtained by stacking the rows of \mathbf{X} . Data exchange among nodes is modeled through graph shift operators, $\mathbf{S}\in\mathbb{R}^{N\times N}$, which respect the sparsity pattern of the graph \mathcal{G} , i.e., $[\mathbf{S}]_{mn}\neq 0$ iff $(m,n)\in\mathcal{E}$ or m=n. Common examples of graph shift operators include the adjacency matrix and the graph Laplacian matrix.

We are given i.i.d. samples of a graph signal $\mathbf{x} \sim q_0(\cdot \mid \mathbf{S}, \mathbf{u})$ supported on a graph \mathbf{S} and additional conditional information represented by another graph signal \mathbf{u} , and we seek to learn a (conditional) generative model $p_{\theta}(\cdot \mid \mathbf{S}, \mathbf{u})$ that approximates $q_0(\cdot \mid \mathbf{S}, \mathbf{u})$.

^{*}This work was partially supported by the Spanish AEI (10.13039/501100011033) grant PID2022-136887NB-I00, and the Community of Madrid via the Ellis Madrid Unit and grants URJC/CAM F1180 and TEC-2024/COM-89.

To this end, we use forward/backward diffusion processes that transport the probability mass between the (conditional) data distribution q_0 and a prior distribution $q_T = \mathcal{N}(\mathbf{0}, \mathbf{I})$ over T diffusion time steps.

The forward diffusion process gradually corrupts data samples \mathbf{x}_0 by adding Gaussian noise at each step. Given an increasing noise schedule $\{\beta_t\}_{t=1}^T$, e.g., linear, the forward diffusion process follows a Markov process with one-step transition densities,

$$q_t(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}). \tag{1}$$

In particular, the process $\mathbf{x}_{1:t}|\mathbf{x}_0$ is Gaussian and so are the conditional densities $q_t(\mathbf{x}_t|\mathbf{x}_0)$ at all steps t. This permits expressing the forward process with a reparametrization [5] as

$$\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$
 (2)

for $t=1,\ldots,T$, where $\alpha_t:=1-\beta_t$ and $\bar{\alpha}_t:=\prod_{i=1}^t\alpha_i$. For sufficiently large diffusion time steps T, the signal converges to isotropic Gaussian noise, i.e., $\mathbf{x}_T\sim\mathcal{N}(\mathbf{0},\mathbf{I})$.

The reverse (backward) diffusion process is also a Markov chain that is a time-reversal of the forward process. That is, the process starts from the prior distribution q_T , terminates at the (conditional) data distribution q_0 and shares the same marginals $\{q_t\}_{t=1}^T$ as the forward process. To this end, the backward diffusion process learns a denoiser, parametrized by θ , that extracts the clean signal \mathbf{x}_0 from \mathbf{x}_t , given the conditional information,

$$\theta^* \in \underset{\theta}{\operatorname{argmin}} \mathbb{E}_{\mathbf{x}_0, t, \mathbf{u}} \| \mathbf{x}_{\theta} (\mathbf{x}_t(\mathbf{x}_0, \epsilon), t; \mathbf{S}, \mathbf{u}) - \mathbf{x}_0 \|^2.$$
 (3)

In (3), the expectation is over the joint distribution of conditional information, graph data samples, and diffusion time steps sampled uniformly in [1,T]. Alternatively, DDPM training [5] learns the parametric function $\epsilon_{\theta^*}(\mathbf{x}_t,t;\mathbf{S},\mathbf{u})$ that approximates the noise ϵ added to \mathbf{x}_0 at each time step t,

$$\boldsymbol{\theta}^* \in \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathbb{E}_{\mathbf{x}_0, t, \mathbf{u}} \left\| \boldsymbol{\epsilon}_{\boldsymbol{\theta}} \left(\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}), t; \mathbf{S}, \mathbf{u} \right) - \boldsymbol{\epsilon} \right\|^2.$$
 (4)

Given an optimal noise predictor ϵ_{θ^*} , the backward diffusion process reverts the forward noising process in (2) at each step by,

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\theta}^*} \left(\mathbf{x}_t, t; \mathbf{S}, \mathbf{u} \right) \right) + \sqrt{\beta_t} \mathbf{w}, \quad (5)$$

where $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Put together, generating novel graph signal samples approximately distributed by the conditional data distribution, i.e., $\mathbf{x}_0 \sim p_{\theta}(\cdot \mid \mathbf{S}, \mathbf{u}) \approx q_0(\cdot \mid \mathbf{S}, \mathbf{u})$, boils down to learning an optimal graph signal denoiser [cf. (3)], or equivalently, an optimal noise predictor [cf. (4)], and running the sampling equation (5) iteratively for $t = T, \ldots, 1$ with $\mathbf{x}_T \sim p_T$. Consequently, we leverage a graph neural network (GNN) architecture for the noise parametrization ϵ_{θ} , which we present next.

3. U-GRAPH NEURAL NETWORKS

We introduce a graph neural network (GNN) architecture for the parametrization of ϵ_{θ} , dubbed U-Graph Neural Network or U-GNN for brevity. A U-GNN with depth B consists of 2B GNN blocks arranged in a U-shape with a left and right path, mirroring convolutional U-Net architectures.

The left path is an encoding (contraction) path and comprises B encoding GNN blocks, which successively extract graph convolutional features and downsample the signals. The right path is a decoding (expansion) path and also consists of B decoding GNN

blocks, which upsample the graph convolutional features and combine them with higher-resolution information provided by skip connections from the left encoding path. Next, we describe a GNN architecture that forms the core of the encoding and decoding blocks.

3.1. Graph Neural Networks

GNNs learn local graph convolutional features by successive applications of graph filters and pointwise nonlinearities [26]. A GNN is a parametrized map Φ that, given a shift operator \mathbf{S} , transforms an input signal $\mathbf{X} \in \mathbb{R}^{N \times F_{\text{in}}}$ into an output signal $\mathbf{V} \in \mathbb{R}^{N \times F_{\text{out}}}$,

$$\mathbf{V} = \mathbf{\Phi}(\mathbf{X}, \mathbf{S}; \mathcal{H}),\tag{6}$$

where \mathcal{H} denotes a set of learnable parameters. The map Φ is a linear composition of L graph convolutional layers, defined as

$$\mathbf{V}_{\ell} = \varphi \left(\sum_{k=0}^{K_{\ell}} \mathbf{S}^{k} \mathbf{V}_{\ell-1} \mathbf{H}_{k,\ell} \right), \tag{7}$$

with $\mathbf{V}_L = \mathbf{V}$. The core component of each layer is a polynomial graph filter that aggregates features over k-hop neighborhoods up to K_ℓ via the learnable parameters $\{\mathbf{H}_{k,\ell}\}_{k=0}^{K_\ell}$. The filter is then followed by a pointwise nonlinear activation function φ , e.g., ReLU. Additional operations such as a normalization layer, e.g., batch or layer normalization, and a pooling operation, e.g., max-pooling, can be subsumed in the definition of φ .

Preceding the first convolutional layer, we incorporate time and conditional embeddings into the input signal **X** via addition and concatenation, respectively. That is, the input to the first graph filter is

$$\mathbf{V}_0 = \left[\mathbf{\Upsilon}_{\mathbf{X}}(\mathbf{X}) + \mathbf{\Upsilon}_{\mathbf{T}}(\mathbf{t}); \, \mathbf{\Upsilon}_{\mathbf{U}}(\mathbf{U}) \right], \tag{8}$$

where $\Upsilon_T : \mathbb{R}^N \mapsto \mathbb{R}^{N \times (F_{\text{in}}/2)}$ is a sinusoidal embedding of vectorized diffusion time steps $\mathbf{t} = t\mathbf{1}_N$. The learnable maps $\Upsilon_X : \mathbb{R}^{N \times F_{\text{in}}} \mapsto \mathbb{R}^{N \times (F_{\text{in}}/2)}$ and $\Upsilon_U : \mathbb{R}^{N \times U} \mapsto \mathbb{R}^{N \times (F_{\text{in}}/2)}$ project the input signal \mathbf{X} and the conditional signal \mathbf{U} into F_{in} -dimensional embeddings, respectively, with parameters included in \mathcal{H} . Note that t and \mathbf{U} are inputs to the GNN, though omitted from the representation in (6) for brevity.

3.2. Encoding & Decoding Blocks

The operation of both encoding and decoding blocks is captured by the GNN defined in (6). Along the encoding path, the F_{b-1} -dimensional feature signal \mathbf{X}_{b-1} , produced by the (b-1)th block, is passed to the bth block to generate

$$\mathbf{X}_b = \mathbf{\Phi}_{\mathrm{E}}(\mathbf{X}_{b-1}, \mathbf{S}; \mathbf{\Xi}_b), \tag{9}$$

where $\mathbf{\Xi}_b$ is the set of learnable parameters of the L-layered GNN in the bth encoding block. The encoding path composes (9) for $b=1,\ldots,B$ to obtain the sequence of graph convolutional features $\{\mathbf{X}_b\}_{b=1}^B$ with reduced dimensionality $F_B < F_{B-1} < \cdots < F_0$. The input feature signal $\mathbf{X}_0 = \mathbf{\Pi}(\mathbf{X})$ is generated by a read-in map $\mathbf{\Pi}: \mathbb{R}^{N \times F} \mapsto \mathbb{R}^{N \times F_0}$, where \mathbf{X} is the matrix representation of the given input $\mathbf{x}_t \in \mathbb{R}^{NF}$ to $\epsilon_{\boldsymbol{\theta}}$.

The decoding path traverses the opposite direction to the encoding path and upsamples the low-dimensional features. The inputs to a decoding block at depth b are the output of the decoding block at depth b+1 and the skip connection from the encoding block at depth b. The decoding block concatenates the inputs in the feature

dimension and processes them with a GNN, similar to the encoding blocks, as

$$\mathbf{Y}_{b-1} = \mathbf{\Phi}_{\mathbf{D}} \Big([\mathbf{Y}_b; \mathbf{X}_b], \, \mathbf{S}; \, \mathbf{\Theta}_b \Big), \tag{10}$$

where Θ_b is the set of learnable parameters of the L-layered GNN in the bth decoding block. For the decoding block at depth B, \mathbf{Y}_B is obtained by a dimension-preserving multilayer perceptron (MLP) transformation of the encoder final output \mathbf{X}_B . The output of the decoding block at depth one \mathbf{Y}_0 is finally mapped by a read-out layer to the model target.

3.3. Down- & Upsampling with Sampling Matrices

The encoder-decoder architecture described so far performs dimensionality reduction only in the feature dimension. However, this need not be the case. Each encoding block can be augmented to pair the GNN with a graph pooling (node downsampling) operator, resulting in output features $\mathbf{X}_b \in \mathbb{R}^{N_b \times F_b}$ for all b, with $N_b \leq N_{b-1}$ and $N_0 = N$ for notational consistency. Similarly, decoding blocks can symmetrically upsample in both node and feature dimensions. Except the following challenge arises after the first encoder block.

The input signal to the first encoder block $\mathbf{X}_0 \in \mathbb{R}^{N_0 \times F_0}$ is already supported on the original graph \mathbf{S} with N_0 nodes, and therefore graph convolutions at this depth are straightforward. However, the subsequent pooling operations leave us with a downsampled node signal $\mathbf{X}_b \in \mathbb{R}^{N_b \times F_b}$, which is lower dimensional than the input signal and is no longer supported on the original graph \mathbf{S} .

To resolve this support mismatch, we treat downsampling as a node selection operation from a predefined subset and leverage *zero padding* to lift downsampled signals back to the original node set. Formally, we represent the node selection mechanism at depth b with a binary fat selection matrix $\mathbf{C}_b \in \{0,1\}^{N_b \times N_{b-1}}$ that satisfies $\mathbf{C}_b \mathbf{1}_{N_{b-1}} = \mathbf{1}_{N_{b-1}}$ and $\mathbf{C}_b^{\top} \mathbf{1}_{N_b} \preceq \mathbf{1}_{N_b}$. Then, the product $\mathbf{C}_b \mathbf{X}_{b-1}$ selects N_b distinct nodes out of the N_{b-1} rows of \mathbf{X}_{b-1} , whereas $\mathbf{C}_b^{\top} \mathbf{X}_b$ lifts \mathbf{X}_b back to the N_{b-1} -node domain by inserting zeroes at the indices of the nodes dropped during downsampling.

More generally, we track the nodes selected from the original graph at depth b by a nested sampling matrix $\mathbf{D}_b \in \{0,1\}^{N_b \times N_0}$,

$$\mathbf{D}_b = \mathbf{C}_b \mathbf{C}_{b-1} \cdots \mathbf{C}_1 = \prod_{i=1}^b \mathbf{C}_i. \tag{11}$$

Here, the product $\widetilde{\mathbf{X}}_b = \mathbf{D}_b^{\top} \mathbf{X}_b$ zero pads \mathbf{X}_b such that the features of the active nodes remain intact at their original indices with respect to \mathbf{S} , while the features of the inactive nodes are set to zero. Hence, the signal $\widetilde{\mathbf{X}}_b$ is supported on the original graph \mathbf{S} , and its convolutions with respect to \mathbf{S} are well-defined.

3.4. Downsampling Encoder & Upsampling Decoder Blocks

To incorporate the downsampling and upsampling into the encoder and decoder blocks, respectively, we redefine the GNN map in (6) to accept the nested selection matrix \mathbf{D} as an additional input,

$$\mathbf{V} = \mathbf{\Psi}(\mathbf{X}, \mathbf{S}, \mathbf{D}; \mathcal{H}). \tag{12}$$

Accordingly, we rewrite the encoder (9) and decoder (10) blocks at a given depth \boldsymbol{b} as

$$\mathbf{X}_b = \mathbf{\Psi}_{\mathrm{E}} (\mathbf{C}_b \mathbf{X}_{b-1}, \mathbf{S}, \mathbf{D}_b; \mathbf{\Xi}_b), \tag{13}$$

$$\mathbf{Y}_{b-1} = \mathbf{\Psi}_{D} \left(\mathbf{C}_{b}^{\top} [\mathbf{Y}_{b}; \mathbf{X}_{b}], \mathbf{S}, \mathbf{D}_{b-1}; \mathbf{\Theta}_{b} \right). \tag{14}$$

Each block uses the selection matrix corresponding to the resolution of its output, which is \mathbf{D}_b for the encoder and \mathbf{D}_{b-1} for the decoder. This incorporation of the nested selection matrix transforms the cornerstone operation of the GNN, i.e., the graph convolution (7), to

$$\mathbf{V}_{\ell} = \varphi \left(\sum_{k=0}^{K_{\ell}} \left[\mathbf{D} (\mathbf{S}^{\gamma})^{k} \mathbf{D}^{\top} \right] \mathbf{V}_{\ell-1} \mathbf{H}_{k,\ell} \right), \tag{15}$$

where γ is a positive integer that determines the stride of the graph shift operation, and its purpose will be clarified shortly. The rest of the GNN architecture remains the same. We note that for $\gamma=1$ and $\mathbf{D}=\mathbf{I}$, (15) coincides with (7).

The convolution in (15) is noteworthy for offering two computationally distinct yet mathematically equivalent viewpoints. The first exploits sampling directly by applying the k reduced shift operators $\mathbf{D}(\mathbf{S}^{\gamma})^k\mathbf{D}^{\top}$ to the subsampled input signal. This approach avoids redundant computations on inactive nodes but requires computing the low-dimensional shift matrices for each k beforehand. The second entails first zero-padding the downsampled signal up to an N-dimensional signal, followed by repeated application of the powers of \mathbf{S} to obtain γk -shifted versions, and lastly subsampling to the desired resolution. We realize this perspective with sparse matrix multiplications for a low computational cost in our implementation.

Finally, the stride parameter γ controls the effective hop size of the graph shift operator. In the zero-padded domain, where $\gamma>1$, \mathbf{S}^{γ} redefines the k-hop aggregation neighborhoods, so that we can choose larger neighborhoods that are effectively intersected with the active nodes selected by \mathbf{D} . This technique remedies the issue of aggregating and pooling mostly zeroes at deeper levels of the U-GNN when zero-padded signals are highly sparse due to successive downsampling operations at higher levels.

4. CASE STUDY: STOCK PRICE FORECASTING

Consider a stock market with N stocks. In stock price forecasting, we are interested in predicting the evolution of log returns over a future horizon of T_h days $\{\mathbf{x}^{(T_p+1)},\ldots,\mathbf{x}^{(T_p+T_h)}\}$, given a history of past observations over the past T_p days $\{\mathbf{u}^{(1)},\ldots,\mathbf{u}^{(T_p)}\}$. Here, $\mathbf{x}^{(k)} \in \mathbb{R}_+^N$ denotes the daily log return ratio of stock prices, whereas $\mathbf{u}^{(k)} \in \mathbb{R}^{N \times U}$ is a collection of U quantities of interest (stock features), including the stock prices themselves, observed on day k.

We are also provided with certain long-term financial indicators (stock fundamentals) such as sector information, market capitalization, trailing price-to-earnings (P/E), profit margins, and so on. From these indicators, we compute a normalized correlation matrix Σ . We model the stock relationship with a graph, and let the weighted adjacency matrix S be the correlation matrix with zeros on the diagonal.

Our goal is to learn a generative model that samples future realizations (trajectories) of log returns $\mathbf{x} = [\mathbf{x}^{(T_p+1)}, \dots, \mathbf{x}^{(T_p+T_h)}]$ given a graph representation \mathbf{S} of stock relationships and a conditional graph signal $\mathbf{u} = [\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(T_p)}]$ pertaining to certain stock features observed over the most recent T_p days. Here we stacked the temporal signals along the feature dimension and wrote them in vector form to match notation with the U-GNN architecture interface.

4.1. Training Details

We experiment on the S&P100 dataset, which includes stocks of N=100 U.S. companies from diverse industries with top market capitalization. We obtain the historical values of the stocks over the last 5 years using the open-source "yfinance" library. Given $T_{\rm p}$ and

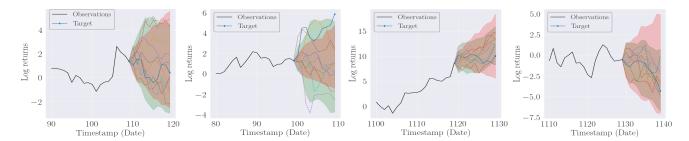


Fig. 1: Ten stochastic log-return forecasts over a $T_h = 10$ -day horizon (dashed), conditioned on a $T_p = 20$ -day history (black) for 4 stocks. For clarity, we plot only 10 trajectories generated by U-GNN. Green and light red shadowed bands cover the 95% confidence intervals for U-GNN and GRW trajectories, respectively. The realized path generally lies within the ensemble of sampled U-GNN trajectories.

Table 1: Comparison of U-GNN against GRW baseline across several configurations and error metrics. A lower value is better for all metrics.

	$T_{\rm p} = 20, T_{\rm h} = 10$				$T_{\rm p} = 20, T_{\rm h} = 20$				$T_{\mathrm{p}}=5, T_{\mathrm{h}}=5$				$T_{\rm p} = 10, T_{\rm h} = 1$			
	MIS	CRPS	RMSE	MAE	MIS	CRPS	RMSE	MAE	MIS	CRPS	RMSE	MAE	MIS	CRPS	RMSE	MAE
U-GNN	20.6	1.09	2.25	1.74	30.8	1.61	3.14	2.28	35.0	1.33	2.20	1.50	23.9	0.85	1.28	0.93
GRW	17.1	1.32	2.39	1.76	30.6	1.98	3.56	2.61	29.7	1.62	2.98	2.08	11.2	0.73	1.29	0.95

 $T_{\rm h}$ values, we split the 5 years period into moderately-sized chunks, shuffled them, and performed a 90 : 5 : 5 split across training, validation, and test datasets. We use the validation dataset solely for early stopping and report the performance on the test dataset.

In our main experiment configuration, the past and future window lengths are $T_{\rm p}=20$ and $T_{\rm h}=10$, respectively. The U-GNN has depth B=3 and each GNN block itself contains L=2 layers with $K_1=K_2=2$ filter taps and the stride is $\gamma=1$. We set the initial number of hidden feature channels to $F_0=64$, which is halved at each depth of the encoding path. We skip the downsampling operation in the first level of U-GNN and downsample twice by a factor of $N_3/N_2=N_2/N_1=0.8$ in the remaining two levels. We adopted a node-degree based selection approach for its simplicity, where we always dropped nodes with the least degree in the graph.

The diffusion process is run for a total of T=500 time steps and a cosine noise schedule $\{\beta_t\}_{t=1}^T$ is used. We train the U-GNN over mini-batches of size 64 for a maximum of 5×10^4 epochs with an AdamW optimizer and an initial learning rate of 2×10^{-2} that decays according to a cosine schedule with warm restarts.

4.2. Evaluation Metrics & Test Performance

In line with the assumption of log-normality of stock prices adopted in most financial models, we benchmark U-GNN against a baseline that models the future evolution of log returns of a stock by a geometric random walk (GRW) whose drift and diffusion coefficients are estimated from the past window.

For evaluation, we sample 20 trajectories of length T_h for each stock from both U-GNN and GRW. We report the root-mean-squared error (RMSE) and mean absolute error (MAE) metrics between the ground truth (target) trajectory and a mean trajectory obtained by ensemble averaging. We also quantify the fidelity of the distribution of trajectories with two probabilistic metrics. Continuous ranked probability score (CRPS) [27] measures the discrepancy between the predicted cumulative distribution function (CDF) of a forecast and a given observation, whereas mean interval score (MIS) evaluates the confidence of a model in its prediction intervals [28].

Fig. 1 shows (cumulative) log-return forecasts for four different stocks over a 10-day horizon conditioned on a 20-day history.

We plot ten trajectories sampled using our trained U-GNN via (5) to capture predictive uncertainty and tail risk. As shown in the figure, the realized path (blue) typically lies within the envelope of the drawn samples and aligns with their dominant direction, indicating the model's ability to produce coherent stochastic scenarios. For example, in the leftmost panel, the conditional window trends very slightly downward for roughly 10 days before tilting up sharply and then down again, and most sample trajectories pivot downward, tracking the ground truth. However, a minority of samples continue an upward move, reflecting residual uncertainty and a nonzero probability of a continued drawup. The remaining panels display less volatile scenarios. Overall, U-GNN balances accuracy with uncertainty robustness better than a GRW baseline.

In Table 1, we report comparative results for four different experiment setups, including the main configuration. We observe that the U-GNN either significantly outperforms or fares similarly to the baseline across all setups and all three evaluation metrics, except for the MIS. We note that this is consistent with the MIS scoring the confidence in prediction intervals, and with the random walk placing the bulk of its probability mass around its mean. The generative model of stock prices might not be as confident in typical scenarios but is capable of capturing rare events and volatilities that are critical to decision-making, as is the case in financial markets. Node pooling (downsampling) did not produce sizable gains in our experiments. We expect more substantial gains on larger graph datasets. Furthermore, better-performing U-GNN models can be trained, specifically for time-series forecasting, by taking into account the spatial and temporal dependencies together, which we leave as future work.

5. CONCLUSION

In this work, we proposed a graph signal generative modeling framework for sampling stochastic graph signals over given graphs and conditional graph signals. We introduced a novel diffusion model architecture termed U-GNN that leverages GNNs as core operational blocks and a graph pooling technique in an effort to generalize convolutional U-Nets. We applied the proposed methodology and U-GNN architecture to a stock price prediction problem. Future work will include a more comprehensive study of the U-GNN architecture.

6. REFERENCES

- [1] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, 2018.
- [2] Weiyu Huang, Antonio G Marques, and Alejandro R Ribeiro, "Rating prediction via graph signal processing," *IEEE Trans. Signal Process.*, vol. 66, no. 19, pp. 5066–5081, 2018.
- [3] Shiwen He et al., "An overview on the application of graph neural networks in wireless networks," *IEEE Open J. Comms. Society*, vol. 2, pp. 2547–2565, 2021.
- [4] Eli Chien et al., "Opportunities and challenges of graph neural networks in electrical engineering," *Nature Reviews Elec. Eng.*, vol. 1, no. 8, pp. 529–546, 2024.
- [5] Jonathan Ho, Ajay Jain, and Pieter Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information* processing systems, vol. 33, pp. 6840–6851, 2020.
- [6] Jiaming Song, Chenlin Meng, and Stefano Ermon, "Denoising diffusion implicit models," in *Intl. Conf. Learning Representa*tions (ICLR), 2021.
- [7] Ling Yang et al., "Diffusion models: A comprehensive survey of methods and applications," ACM Comp. Surveys, vol. 56, no. 4, pp. 1–39, 2023.
- [8] Hanqun Cao et al., "A survey on generative diffusion models," IEEE Trans. Knowledge Data Engineering, vol. 36, no. 7, pp. 2814–2830, 2024.
- [9] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang, "Score-based generative modeling of graphs via the system of stochastic differential equations," in *Intl. Conf. Learning Representations* (ICLR). PMLR, 2022, pp. 10362–10383.
- [10] Clément Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard, "Digress: Discrete denoising diffusion for graph generation," in *Intl. Conf. Learning Representations (ICLR)*, 2023.
- [11] Ruixin Chen, Jianping Fan, Meiqin Wu, Rui Cheng, and Jiawen Song, "G-Diff: A graph-based decoding network for diffusion recommender model," *IEEE Trans. Neural Netw. Learning Syst.*, 2024.
- [12] Yunqin Zhu, Chao Wang, Qi Zhang, and Hui Xiong, "Graph signal diffusion model for collaborative filtering," in ACM SI-GIR Conf. Research Dev. Inform. Retrieval, 2024, pp. 1380– 1390.
- [13] Jujia Zhao, Wang Wenjie, Yiyan Xu, Teng Sun, Fuli Feng, and Tat-Seng Chua, "Denoising diffusion recommender model," in ACM SIGIR Conf. Research Dev. Inform. Retrieval, 2024, pp. 1370–1379.
- [14] Haomin Wen et al., "DiffSTG: Probabilistic spatio-temporal graph forecasting with denoising diffusion models," in *Proceedings of the 31st ACM international conference on advances in geographic information systems*, 2023, pp. 1–12.

- [15] Huipeng Zhang, Honghui Dong, and Zhiqiang Yang, "TS-GDiff: Traffic state generative diffusion model using multi-source information fusion," *Transportation Research Part C: Emerging Technologies*, vol. 174, pp. 105081, 2025.
- [16] Junfeng Hu, Xu Liu, Zhencheng Fan, Yuxuan Liang, and Roger Zimmermann, "Towards unifying diffusion models for probabilistic spatio-temporal graph learning," in ACM Int. Conf. Advances in Geographic Inf. Systems, 2024, pp. 135–146.
- [17] Divyanshu Daiya, Monika Yadav, and Harshit Singh Rao, "Diffstock: Probabilistic relational stock market predictions using diffusion models," in *IEEE Intl. Conf. Acoust., Speech Signal Process. (ICASSP)*. IEEE, 2024, pp. 7335–7339.
- [18] Zinuo Henry You, Zijian Shi, Hongbo Bo, John P Cartlidge, Li Zhang, and Yan Ge, "DGDNN: Decoupled graph diffusion neural network for stock movement prediction," in *Intl. Conf. Agents Artificial Intel.* SciTePress, 2024, pp. 431–442.
- [19] Mehdi Letafati, Samad Ali, and Matti Latva-Aho, "Conditional denoising diffusion probabilistic models for data reconstruction enhancement in wireless communications," *IEEE Trans. Machine Learning Comms. Net.*, 2024.
- [20] Yigit Berkay Uslu, Samar Hadou, Shirin Saeedi Bidokhti, and Alejandro Ribeiro, "Generative diffusion models for resource allocation in wireless networks," arXiv preprint arXiv:2504.20277, 2025.
- [21] Nuowen Kan, Sa Yan, Junni Zou, Wenrui Dai, Xing Gao, Chenglin Li, and Hongkai Xiong, "Gdplan: Generative network planning via graph diffusion model," *IEEE Trans. Netw.*, 2025.
- [22] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro, "Graphs, convolutions, and neural networks: From graph filters to graph neural networks," *IEEE Signal Process*ing Magazine, vol. 37, no. 6, pp. 128–138, 2020.
- [23] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro, "Graph neural networks: Architectures, stability, and transferability," *Proc. IEEE*, vol. 109, no. 5, pp. 660–682, 2021.
- [24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Intl. Conf. on Medical Image Comp. and Comp.-Assisted Int.* Springer, 2015, pp. 234–241.
- [25] Yiyong Feng, Daniel P Palomar, et al., "A signal processing perspective on financial engineering," *Foundations and Trends*® *in Signal Processing*, vol. 9, no. 1–2, pp. 1–231, 2016.
- [26] Fernando Gama, Antonio G Marques, Geert Leus, and Alejandro Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Transactions on Signal Processing*, vol. 67, no. 4, pp. 1034–1049, 2018.
- [27] James E Matheson and Robert L Winkler, "Scoring rules for continuous probability distributions," *Management science*, vol. 22, no. 10, pp. 1087–1096, 1976.
- [28] Tilmann Gneiting and Adrian E Raftery, "Strictly proper scoring rules, prediction, and estimation," *Journal of the American Statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.