NEW WAYS TO CONSTRUCT GRAPH NEURAL NETWORKS FROM VARIATIONAL MODELS AND CONTROL APPROACHES

LINGFENG LI*, HAO LIU †, XUE-CHENG TAI ‡, AND RAYMOND HON-FU CHAN§

Abstract. Graph neural networks (GNNs) have recently gained significant research attention and found applications across diverse domains. Unlike traditional convolutional neural networks (CNNs), which operate on structured grids, GNNs are defined on unstructured graphs. Despite their wide applications, there remains a lack of work providing mathematical interpretations of GNNs. In this paper, we propose a novel framework for designing and constructing GNN models from graphon control problems, where graphon is the large graph limit of graph models. We also demonstrate that many classical message-passing GNN architectures can be interpreted via our framework. Motivated by this connection, we formulate a novel control problem based on the gradient flow of the multiphase Potts model. By discretizing this control problem using a specific operator-splitting scheme and employing the unrolling technique, we construct novel GNN architectures called GraphPottsNets (GPN). The intrinsic regularization properties of the Potts model are preserved in the proposed GPNs, providing a regularization effect during forward propagation of graph signals. Comprehensive experiments demonstrate that our GPNs achieve competitive results compared to state-of-the-art GNN models across multiple benchmarks. Moreover, our GPNs exhibit significantly greater robustness against node feature perturbations and maintain high prediction accuracy even under intense noise on node features.

Key words. graph neural networks; Potts model; operator-splitting

1. Introduction.

There has been a recent surge of interest in Graph Neural Networks (GNNs), driven by their ability to model relational data and capture complex dependencies in unstructured systems. These models have found wide applications across diverse scientific domains, including chemistry, physics, and neuroscience [15, 21, 13, 43, 23]. Though GNN models achieved superior performances in various tasks, they are shown to be vulnerable to adversarial attacks and noises [59, 20, 47, 60]. In some safety-critical and high-stakes applications, like drug discovery and financial fraud detection, robustness is a highly valued property. Real-world graph data often contains inherent noise due to measurement errors, incomplete information, or label inaccuracies. That is why developing robust GNNs has become an urgent priority across many fields.

One important class of GNN is the message-passing GNN, which operates by passing messages between the nodes of a graph, allowing information to propagate throughout the network. In a message-passing GNN, each node is associated with a feature vector representing its characteristics or attributes. The network iteratively updates these feature vectors based on the information received from other nodes. This update process typically involves aggregating and transforming the messages received from neighboring nodes and then combining them with the node's own features to produce an updated representation. This mechanism allows the network to capture relational dependencies and structural information contained in the graph. By iteratively passing messages and updating node representations, the network can learn to incorporate information from the entire graph into specific tasks. Popular message-passing GNNs include the Graph Convolutional Network (GCN) [25], GraphSAGE [24], Graph Attention Network (GAT) [52], Graph Transformer (GT) [16], and Graph Isomorphism Network (GIN) [55].

Besides the GNNs-based methods, energy-based methods are also commonly used to solve graph

*Hetao Institute of Mathematics and Interdisciplinary Sciences (Shenzhen), Shenzhen, China.
†Department of Mathematics, Hong Kong Baptist University, Hong Kong
‡Norwegian Research Center, Bergen, Norway
§Department of Data Science, and Department of Operations and Risk Management, Lingnan University, Hong Kong; Hong Kong Centre for Cerebro-Cardiovascular Health Engineering, Hong Kong.

applications, especially node classification tasks [3, 4, 19, 5, 45]. This type of method operates by minimizing a problem-dependent energy function. The graph cut algorithm [12], which has been extensively applied in various fields, solves a node partition problem by finding a cut with minimum cost. The graph MBO method [35] generalizes the Ginzburg-Landau energy to graphs and applies the MBO scheme, which consists of a diffusion step and a threshold step, to minimize it. The Potts model has also been used for graph tasks [57] with different region force terms, in which the nodes are partitioned by minimizing the Potts energy function.

Recently, there have been many works focusing on connecting Convolutional Neural Networks (CNNs), which are usually defined on structured data, with continuous control problems and variational models [46, 26, 48, 29, 54, 50]. The multi-layer structure of neural networks can be explained as some types of iterative numerical schemes for solving optimization problems or control problems. By leveraging this continuous point of view, we can modify and design new network structures by incorporating prior knowledge as regularization terms into the variational models or control problems, and obtain networks that can preserve some desirable properties. For example, in [48, 29], the authors proposed novel network structures based on continuous control problems derived from the Potts model, which contains a boundary regularization term. In particular, the regularization term explicitly imposes some structural constraints on the feature and therefore improves the robustness. The model can achieve high accuracy when the network input is disturbed by high level of Gaussian noise, and outperforms the current state-of-the-art models.

Graph models are typically defined on unstructured data, which lacks an explicit continuous counterpart. However, we can define a continuous graph via the limit of a sequence of graphs with increasing size. The continuous graph defined in this way is called a graphon [6, 7]. A graphon is a measurable function defined on a bounded set, and can be viewed as an undirected graph with an uncountable number of nodes. The theory of graphons and its connection with graph models have been extensively studied [6, 7, 32, 58]. Graphons have also been used for the analysis of GNN in the literature. For example, Ruiz *et al.* [41, 42] defined the graphon neural network (called WNN) as the limiting object of a sequence of GNNs, and used the defined WNNs to analyze the transferability and stability of GNNs. Levie [28] also used the continuous graphon to study the generalization error and stability of message-passing GNNs.

In this work, we aim to extend the continuous viewpoint of CNNs to GNNs. We will build connections between the message-passing structures of GNNs and control problems defined for graphon functions. This graphon viewpoint is primarily used as a theoretical framework for deriving the proposed network architectures and interpreting existing GNN models. Following the ideas of a series of previous works [30, 29, 48, 49], we will provide a unified framework that can interpret classical message-passing structures of GNN models as operator-splitting of certain graphon control problems. Based on this framework, we will propose robust GNN models by introducing extra regularization and control variables to the control problems.

We emphasize that our approach is fundamentally different from algorithm "unfolding" (also called unrolling), where an iterative optimization algorithm is transformed into a trainable neural network architecture. In such approaches, each iteration of the algorithm becomes one network layer (or block), and certain algorithmic parameters are learned from data [22, 56]. In contrast, our framework is based on the design of a continuous-time control problem, in which the control variables become the training parameters after discretization. For several classes of problems, approximate controllability and universal approximation properties have been established [11].

This work is organized as follows: Section 2 introduces some basic notations of graph models and graph neural networks. Section 3 builds the connections between the graphon Potts model and

control problems, for which the operator-splitting strategies are introduced and their connections with existing state-of-the-art GNN models are discussed in Section 4. We then propose a new GNN, called GraphPottsNet, based on the splitting schemes for the graph Potts model in Section 5. We test our GraphPottsNet with systematic experiments in Section 6. This paper is concluded in Section 7.

2. Preliminaries. In this section, we introduce basic concepts of graph models, and how operator-splitting methods can be used to solve them.

2.1. Graph models and large-graph limit. Consider an undirected graph with N nodes: $\mathcal{G} = (V, \mathcal{E})$ where $V = \{v_i\}_{i=1}^N$ is the set of nodes and $\mathcal{E} \subset V^2$ is the set of edges. Each edge $(v_i, v_j) \in \mathcal{E}$ is assigned a positive weight $\omega_{ij} > 0$. We extend the definition of weights to non-edge pairs by $\omega_{ij} = 0$ if $(v_i, v_j) \notin \mathcal{E}$. Define a weight matrix $W_N \in \mathbb{R}^{N\times N}$ such that $(W_N)_{ij} = \omega_{ij}$ for $i, j \in [N]$ where $[N] = \{1, 2, \ldots, N\}$. The structure of $\mathcal{G}$ can be precisely characterized by W_N. For each $i \in [N]$, the degree of the i-th node is defined as $d_i := \sum_{j\in[N]} \omega_{ij}$. We make some further assumptions on the graph model $\mathcal{G}$ considered in this paper:

1. $\mathcal{G}$ is undirected: $\omega_{ij} = \omega_{ji}$ for all $(v_i, v_j) \in \mathcal{E}$, which means W_N is symmetric.
2. $\mathcal{G}$ contains no self-loops, i.e., all diagonal elements of W_N are 0.
3. $\mathcal{G}$ contains no isolated points, i.e., $d_i > 0$ for all $i \in [N]$.

A graphon is a bounded, symmetric, measurable function $\mathcal{W} : [0, 1)^2 \to \mathbb{R}$ that can be viewed as an undirected graph with an uncountable number of nodes. Every $x \in [0, 1)$ represents a node, $\mathcal{W}(x, y)$ represents the weight between two nodes x and y, and $d(x) = \int_0^1 \mathcal{W}(x, y)dy$ represents the degree of node x. The graph $\mathcal{G}$ with weight matrix W_N can be associated with a piecewise constant graphon $\mathcal{W}_N(x, y)$ defined on $[0, 1)^2$ by

$$\mathcal{W}_N(x, y) = \omega_{ij}, \quad (x, y) \in I_i \times I_j, i, j \in [N]$$

where $I_i = [\frac{i-1}{N}, \frac{i}{N})$ for $i \in [N]$. For any graphon $\mathcal{W} : [0, 1)^2 \to \mathbb{R}$, we can define its cut norm as

$$\|\mathcal{W}\|_\square := \sup_{S_1,S_2\subseteq[0,1)} \left| \int_{S_1\times S_2} \mathcal{W}(x, y)dxdy \right|$$

where the supremum is taken over all measurable subsets S_1 and S_2 of $[0, 1)$. When $\mathcal{W}$ is always non-negative or non-positive, the supremum is obtained simply when $S_1 = S_2 = [0, 1)$.

For any sequence of graphs $\{W_N\}_{N=1}^\infty$ and associated piecewise constant graphons $\{\mathcal{W}_N\}_{N=1}^\infty$, if there exists a graphon $\mathcal{W}$ such that the cut norm of $\mathcal{W}_N - \mathcal{W}$ converges to 0 as $N \to \infty$, we say this sequence of graphs $\{W_N\}$ converges to $\mathcal{W}$ [6]. We notice that $\mathcal{W}_N - \mathcal{W}$ may contain both negative and positive parts, so the supremum of the cut norm is achieved by choosing non-trivial S_1 and S_2. It has been shown that every graphon function is the limiting object of a convergent graph sequence, and every convergent graph sequence has a limit graphon [32, 6, 7]. Each graph weight matrix W_N can be viewed as a discretization of a continuous graphon $\mathcal{W}$ on an N by N uniform grid.

The connections between continuous graphons and finite graphs have been extensively studied [6, 7]. It has been proved that some energy functionals for graphons, such as the cut functional, total variation, and Ginzburg-Landau functional, are the Γ-limit of sequences of finite graph functionals [8, 58, 51]. We present some details about the definition of operators on graphs and graphons in Appendix A.

2.2. The graphon Potts model and gradient flows. The graphon Potts model for binary classification of nodes x is formulated as

$$\min_{u(x)\in\{0,1\}} \frac{\alpha}{2} \underbrace{\mathrm{TV}_{\mathcal{W}}(u)}_{\text{Total variation term}} + \underbrace{\langle u, g\rangle_{\mathcal{W}}}_{\text{Data fidelity term}}, \tag{2.1}$$

where u is the partition function, g is the given region force function, and α is the weight of the total variation term. The definition of the graphon total variation $\mathrm{TV}_{\mathcal{W}}$ and inner product $\langle\cdot,\cdot\rangle_{\mathcal{W}}$ can be found in Appendix A (see (A.6) and (A.7)). The total variation measures the length of the partition boundary, and the data fidelity term measures the consistency with known data. Problem (2.1) is the continuous counterpart of the graph Potts model. More details about the connection between the variational models on graphs and graphons can be found in Appendix B. By solving (2.1), each $x \in [0,1)$ can be classified into two classes based on the value of $u(x)$. Model (2.1) can be easily extended to a multi-phase classification problem, where we denote the number of classes as c. Let $U(x) \in \Delta^c$ be a partition function, where

$$\Delta^c := \left\{ (u_1(x), \dots, u_c(x)) \in \{0,1\}^c \,\middle|\, \sum_{k=1}^{c} u_k(x) = 1, \forall x \in [0,1) \right\},$$

and $G(x) = (g_1(x), \dots, g_c(x)) \in \mathbb{R}^c$ be a multiphase region force function. The multi-phase Potts model is given as:

$$\min_{U\in\Delta^c} \frac{\alpha}{2} \sum_{k=1}^{c} \mathrm{TV}_{\mathcal{W}}(u_k) + \sum_{k=1}^{c} \langle u_k, g_k\rangle_{\mathcal{W}}. \tag{2.2}$$

Because of the nonsmoothness of the total variation term and the nonconvexity of the constrained set Δ^c, it is difficult to directly solve model (2.2), and some relaxation techniques, such as the soft threshold dynamics approach [31], can be used. We relax the model (2.2) as (see Appendix C for detailed derivation):

$$\min_{U\in\widetilde{\Delta}^c} \frac{\alpha}{2} \sum_{k=1}^{c} \langle u_k, \Delta_{\mathcal{W}} u_k\rangle_{\mathcal{W}} + \sum_{k=1}^{c} \langle u_k, g_k\rangle_{\mathcal{W}} + \beta \sum_{k=1}^{c} \langle u_k, \log(u_k)\rangle_{\mathcal{W}} \tag{2.3}$$

where

$$\widetilde{\Delta}^c := \left\{ (u_1(x), \dots, u_c(x)) \in [0,1]^c \,\middle|\, \sum_{k=1}^{c} u_k(x) = 1, \forall x \in [0,1) \right\},$$

$\Delta_{\mathcal{W}}$ is the graphon Laplacian as defined in (A.4), and β is the weight of the entropy term. The relaxed model (2.3) can be solved through the gradient flow:

$$\frac{\partial U}{\partial t}(x,t) = -\underbrace{G(x)}_{\text{Region force}} \underbrace{-\alpha\Delta_{\mathcal{W}} U(x,t)}_{\text{Diffusion term}} + \underbrace{\mathcal{S}(U)(x,t)}_{\text{Nonlinear term}}, \tag{2.4}$$

where $\mathcal{S}(U) = -\beta - \beta\log(U) - \partial I_{\widetilde{\Delta}^c}(U)$, $\Delta_{\mathcal{W}} U = (\Delta_{\mathcal{W}} u_1, \dots, \Delta_{\mathcal{W}} u_c)$, $I_{\widetilde{\Delta}^c}$ is the indicator function of the set $\widetilde{\Delta}^c$, and $\partial I_{\widetilde{\Delta}^c}$ denotes its subgradient.

The gradient flow of the Potts model can be further used to construct novel neural network models. In [48, 29], the authors discretize a control problem, which is modified from the gradient

flow of the Potts model for binary image segmentation, using carefully designed operator-splitting algorithms and construct novel neural networks by the unrolling method. Because of the existence of regularization terms such as the total variation in the Potts model, the unrolled network is more robust to noise in the input space than standard neural network models.

3. The proposed graphon control problems. Inspired by [48, 29], we derive a general framework to design graph neural networks based on the gradient flow (2.4). Suppose we are given a graphon signal f, which is a function defined on $[0,1)$. We are interested in modeling a solution operator $\mathcal{T}$ that maps f to the desired output space, which depends on the nature of the task. We consider modeling the solution operator $\mathcal{T}$ through the following model:

$$\begin{cases} \frac{\partial U}{\partial t} = -G - \alpha \Delta_{\mathcal{W}} U + \mathcal{A}(U,t) + \mathcal{S}(U) + \hat{\mathcal{S}}(U,t) & (x,t) \in [0,1) \times [0,T], \\ U(x,0) = \mathcal{U}_0(x;f) & x \in [0,1), \\ \mathcal{T}(f) = \mathcal{H}(U_T), \end{cases} \tag{3.1}$$

where $U(x) \in \mathbb{R}^K$ is the hidden state variable, K is the hidden dimension, $\mathcal{U}_0(x;f)$ is the initial condition derived from the input signal f, $G(x) \in \mathbb{R}^K$ is the region force in the hidden space, $\mathcal{A}(U,t)(x,t) \in \mathbb{R}^K$ is a graphon convolution function, $\mathcal{S}(U) \in \mathbb{R}^K$ is a nonlinear function, and $\hat{\mathcal{S}}(U,t) \in \mathbb{R}^K$ is a time-dependent nonlinear function. The final output $\mathcal{T}(f)$ is computed by a transformation operator $\mathcal{H}$ acting on U_T, which is U at the terminal time $t = T$. The operator $\mathcal{H}$ serves as a post-processing operator, and the choice of $\mathcal{H}$ is task-dependent. For example, in a graph classification problem with c classes, $\mathcal{H}$ can be defined as $\mathcal{H}(U_T) := \int_{[0,1)} Z U_T(x) dx \in \mathbb{R}^c$, where $Z \in \mathbb{R}^{c \times K}$ is a transform matrix. Similar to [48, 29], in (3.1), we introduce an extra linear convolution term $\mathcal{A}$ to the Potts gradient flow (2.4) to better control the evolutionary behavior of the state variable U. Besides, we also introduce the time-dependent nonlinear term $\hat{\mathcal{S}}$, which will allow a more flexible design of the activation function when constructing neural networks. The diffusion term $\Delta_{\mathcal{W}} U$, which corresponds to the derivative of the total variation term, ensures that U assigns similar hidden-state values to neighboring nodes $x \in [0,1)$, thus promoting spatial consistency in U. The weight α modulates the strength of this regularization, with larger α encouraging higher spatial consistency. We also remark that the nonlinear term $\mathcal{S}(U)$ in (3.1) can either coincide with $\mathcal{S}(U)$ in (2.4) or represent a different nonlinear term designed for specific tasks. In the following section, we propose operator-splitting methods to solve (3.1), and show that the resulting scheme has a similar architecture to graph neural networks.

The control problem (3.1) is motivated by the graphon Potts gradient flow (2.4). Specifically, if we choose K to be the number of classes c in the Potts model (2.4), then the evolution of U in (3.1) is equivalent to the Potts model gradient flow (2.4) with an extra linear convolution term. However, (3.1) is proposed here as a general hidden-state evolution model and the output dimension is determined by the task through the post-processing operator $\mathcal{H}$, not by the hidden dimension K. For example, in graph regression, $\mathcal{T}(f) \in \mathbb{R}$ directly predicts the value of the regression target, and $\mathcal{H}$ maps U_T to a real value. Here, the parameter K denotes the hidden dimension of the state variable U, so it plays the same role as the hidden width in a standard neural network. Increasing K enlarges the feature space in which the graph dynamics evolve and therefore increases the model capacity. Though it may cause more computational cost, a larger hidden dimension generally allows the network to represent more complex mappings and therefore achieve better results on specific tasks. Theoretical results regarding the hidden width and model capacity can be found, for example, in [33]. In practice, we can choose a proper K according to the computational budget. In the rest of this paper, we consider the general form of (3.1), where $U(x) \in \mathbb{R}^K$ is a hidden state variable

and K indicates the dimension of the hidden space. The operator $\mathcal{H}$ will then transform U_T into the desired output space according to different tasks. We give examples of $\mathcal{H}$ for different tasks in Appendix D.

Based on (3.1), we consider using an optimal control approach to obtain the solution operator $\mathcal{T}(f)$. More specifically, we assume there is a set of graphon signals $f_{\hat{i}}$ and the corresponding target quantity $v_{\hat{i}}$ for $\hat{i} = 1, \ldots, \hat{N}$, where $\hat{N}$ denotes the total number of training samples. We denote the mapping from $f_{\hat{i}}$ to $\mathcal{T}(f_{\hat{i}})$ via (3.1) as $\mathcal{N}_\zeta$ where $\zeta = \{\mathcal{U}_0, G, \mathcal{A}, \hat{\mathcal{S}}, \mathcal{H}\}$ represents the collection of all undetermined variables in (3.1). The optimal control problem can be formulated as

$$\min_{\zeta} \sum_{\hat{i}=1}^{\hat{N}} L(\mathcal{N}_\zeta(f_{\hat{i}}), v_{\hat{i}}) \tag{3.2}$$

where L is a task-dependent loss function measuring the difference between $\mathcal{N}_\zeta(f_{\hat{i}})$ and $v_{\hat{i}}$. For example, L can be the cross-entropy function for classification tasks or it can be the squared L_2 error for regression tasks.

4. Proposed operator-splitting scheme for (3.1). In this work, we consider $\mathcal{A}$ in (3.1) as

$$\mathcal{A}(U,t) = \left(\sum_{s=1}^{K} \mathbf{a}_{1,s}(t) *_{\hat{\mathcal{W}}} u_s + b_1(t), \sum_{s=1}^{K} \mathbf{a}_{2,s}(t) *_{\hat{\mathcal{W}}} u_s + b_2(t), \ldots, \sum_{s=1}^{K} \mathbf{a}_{K,s}(t) *_{\hat{\mathcal{W}}} u_s + b_K(t) \right),$$

where $*_{\hat{\mathcal{W}}}$ is the graphon convolution defined as (A.5), $\mathbf{a}_{k,s}(t) \in \mathbb{R}^2$ is a convolution kernel, and $\mathbf{b}(t) = (b_1(t), \ldots, b_K(t)) \in \mathbb{R}^K$ is the bias term. Here, the function $\hat{\mathcal{W}}(x,y)$ can be chosen as the graphon function $\mathcal{W}(x,y)$ or its normalized version, like $\mathcal{W}(x,y)/d(x)$ and $\mathcal{W}(x,y)/\sqrt{d(x)d(y)}$.

We first discretize the time domain $[0,T]$ using a mesh $\{\tau h\}_{\tau=0}^{M}$ where $h = T/M$. Then, we apply a simple sequential splitting Algorithm E.1 discussed in Appendix E (we refer to [48, Appendix C] for an introduction to general operator-splitting algorithms) directly to (3.1), which leads to the following time-discretizations:

$$U^\tau = (\mathcal{I} - h\mathcal{S})^{-1} \left(U^{\tau-1} + h \left(-G - \alpha \Delta_{\mathcal{W}} U^{\tau-1} + \mathcal{A}^{\tau-1}(U^{\tau-1}) + \hat{\mathcal{S}}^{\tau-1}(U^{\tau-1}) \right) \right). \tag{4.1}$$

Here, we use the superscript τ to denote a time-dependent variable discretized at the τ-th time step. The optimal control problem (3.2) can be approximated by

$$\min_{\zeta_M} \sum_{\hat{i}=1}^{\hat{N}} L(\mathcal{N}_{\zeta_M}(f_{\hat{i}}), v_{\hat{i}}), \tag{4.2}$$

where $\mathcal{N}_{\zeta_M}(f_{\hat{i}})$ denotes $\mathcal{H}(U^M)$ with U^M being computed iteratively using specific numerical schemes from $U^0 = \mathcal{U}_0(x; f_{\hat{i}})$, and $\zeta_M = \left\{ \mathcal{U}_0, G, \{\{\mathbf{a}_{s,k}^{\tau-1}\}_{s,k=1}^{K}, \mathbf{b}^{\tau-1}\}_{\tau=1}^{M}, \{\hat{\mathcal{S}}^{\tau-1}\}_{\tau=1}^{M}, \mathcal{H} \right\}$ denotes the collection of all undetermined variables discretized at each time step.

By the definitions in Appendix A, we write $\mathbf{a}_{k,s}(t) = (a_{k,s,0}(t), a_{k,s,1}(t))$. Then the k-th component of $\mathcal{A}$ is

$$\mathcal{A}_k(U,t)(x) = \sum_{s=1}^{K} \left(a_{k,s,0}(t) u_s(x) + a_{k,s,1}(t) \int_0^1 \hat{\mathcal{W}}(x,y) u_s(y)\, dy \right) + b_k(t),$$

while

$$-\alpha\Delta_{\mathcal{W}}u_k(x) = -\alpha\left(u_k(x) - \frac{1}{d(x)}\int_0^1 \mathcal{W}(x,y)u_k(y)\,dy\right).$$

If we choose $\hat{\mathcal{W}}(x,y) = \mathcal{W}(x,y)/d(x)$, then the k-th component of $\mathcal{A}(U,t) - \alpha\Delta_{\mathcal{W}}U$ can be written as

$$\sum_{s=1}^{K}\left((a_{k,s,0}(t) - \alpha\delta_{k,s})u_s(x) + (a_{k,s,1}(t) + \alpha\delta_{k,s})\int_0^1 \hat{\mathcal{W}}(x,y)u_s(y)\,dy\right) + b_k(t)$$

where $\delta_{k,s} = 1$ if $k = s$ and $\delta_{k,s} = 0$ otherwise. If we compute U^M directly through (4.1), then the optimal value of the training problem (4.2) is independent of α. To see this, assume first that (4.2) attains its minimum when $\alpha = \alpha_0$, with coefficients $\mathbf{a}_{k,s}(t) = \mathbf{a}^*_{k,s}(t) = (a^*_{k,s,0}(t), a^*_{k,s,1}(t))$ for each k and s. If we then change α to another value α_1, we define a new set of coefficients by taking $\mathbf{a}_{k,s}(t) = \mathbf{a}^*_{k,s}(t)$ for $k \neq s$ and

$$\mathbf{a}_{k,k}(t) = \left(a^*_{k,k,0}(t) - \alpha_1 + \alpha_0,\ a^*_{k,k,1}(t) + \alpha_1 - \alpha_0\right)$$

for each k, while keeping all other undetermined variables unchanged. Then the training problem (4.2) attains the same minimum with this new set of coefficients, and the resulting trajectory U^τ for every training sample is exactly the same as the one obtained from $\mathbf{a}^*_{k,s}(t)$ when $\alpha = \alpha_0$. Therefore, the optimal value of the training problem (4.2) is independent of α, and the effect of α may be compensated by the learnable parameters $\mathbf{a}_{k,s}(t)$.

To make the role of α more explicit, we design two hybrid operator-splitting algorithms to solve (3.1). We first split operators in (3.1) into two groups: $-G(x) - \alpha\Delta_{\mathcal{W}}U + \mathcal{A}(U,t)(x,t) + \hat{\mathcal{S}}(U,t)$ and $\mathcal{S}(U)$ sequentially as Algorithm E.1. Then, the first group of operators is further split into two subgroups: $-G(x) - \alpha\Delta_{\mathcal{W}}U(x,t)$ and $\mathcal{A}(U,t)(x,t) + \hat{\mathcal{S}}(U,t)$. These two subgroups can be split by either a sequential scheme or a parallel scheme. Both methods are efficient and have the same order of accuracy (See Appendix F). Consequently, we will consider both sequential and parallel splitting schemes for this group of operators, resulting in two different hybrid splitting algorithms as presented in the following subsections. In the proposed splitting schemes, the two operator-splitting steps can be different, either sequential or parallel, and the temporal discretization can be different, either implicit or explicit. This is the reason why we call the splitting scheme "hybrid". We will discuss the details of the two splitting schemes in the following subsections.

4.1. The first splitting algorithm for (3.1). We first consider the parallel splitting of $-G(x) - \alpha\Delta_{\mathcal{W}}U(x,t)$ and $\mathcal{A}(U,t)(x,t) + \hat{\mathcal{S}}(U,t)$, as shown in Algorithm 4.1.

In Algorithm 4.1, we still need to solve three subproblems. We solve (4.3) as:

$$\begin{aligned} U^{\tau-\frac{1}{2},1} &= (\mathcal{I} - 2h\hat{\mathcal{S}}^\tau)^{-1}\left(U^{\tau-1} + 2h\mathcal{A}^{\tau-1}(U^{\tau-1})\right) \\ &= (\mathcal{I} - 2h\hat{\mathcal{S}}^\tau)^{-1}\begin{pmatrix} u_1^{\tau-1} + 2h\sum_{s=1}^K \mathbf{a}_{1,s}^{\tau-1} *_{\hat{\mathcal{W}}} u_s^{\tau-1} + 2hb_1^{\tau-1} \\ u_2^{\tau-1} + 2h\sum_{s=1}^K \mathbf{a}_{2,s}^{\tau-1} *_{\hat{\mathcal{W}}} u_s^{\tau-1} + 2hb_2^{\tau-1} \\ \vdots \\ u_K^{\tau-1} + 2h\sum_{s=1}^K \mathbf{a}_{K,s}^{\tau-1} *_{\hat{\mathcal{W}}} u_s^{\tau-1} + 2hb_K^{\tau-1} \end{pmatrix}. \end{aligned}$$

Algorithm 4.1 The first hybrid splitting method

Input: The initial condition $U(x,0)$.
Output: The computed solution $U(x,T)$.
Set $U^0 = U(x,0)$.
for $\tau = 1, \cdots, M$ **do**
Solve for $U^{\tau-\frac{1}{2},1}$:

$$\frac{U^{\tau-\frac{1}{2},1} - U^{\tau-1}}{2h} = \mathcal{A}^{\tau-1}(U^{\tau-1}) + \hat{\mathcal{S}}^{\tau}(U^{\tau-\frac{1}{2},1}) \tag{4.3}$$

Solve for $U^{\tau-\frac{1}{2},2}$:

$$\frac{U^{\tau-\frac{1}{2},2} - U^{\tau-1}}{2h} = -G - \alpha\Delta_{\mathcal{W}}U^{\tau-1} \tag{4.4}$$

Combine $U^{\tau-\frac{1}{2},1}$ and $U^{\tau-\frac{1}{2},2}$:

$$U^{\tau-\frac{1}{2}} = \frac{1}{2}(U^{\tau-\frac{1}{2},1} + U^{\tau-\frac{1}{2},2})$$

Solve for U^{τ}:

$$\frac{U^{\tau} - U^{\tau-\frac{1}{2}}}{h} \in \mathcal{S}(U^{\tau}) \tag{4.5}$$

end for

Subproblem (4.4) is solved explicitly:

$$U^{\tau-\frac{1}{2},2} = U^{\tau-1} + 2h(-G - \alpha\Delta_{\mathcal{W}}U^{\tau-1}) = \begin{pmatrix} u_1^{\tau-1} + 2h(-g_1 - \alpha\Delta_{\mathcal{W}}u_1^{\tau-1}) \\ u_2^{\tau-1} + 2h(-g_2 - \alpha\Delta_{\mathcal{W}}u_2^{\tau-1}) \\ \vdots \\ u_K^{\tau-1} + 2h(-g_K - \alpha\Delta_{\mathcal{W}}u_K^{\tau-1}) \end{pmatrix}.$$

The last substep (4.5) is solved implicitly:

$$U^{\tau} = (\mathcal{I} - h\mathcal{S})^{-1}U^{\tau-1/2}.$$

We formulate the full update from $U^{\tau-1}$ to U^{τ} as:

$$U^{\tau} = \hat{\sigma}_1\left(\frac{1}{2}U^{\tau-1} + h(-G - \alpha\Delta_{\mathcal{W}}U^{\tau-1}) + \frac{1}{2}\hat{\sigma}_2^{\tau}\left(U^{\tau-1} + 2h\mathcal{A}^{\tau-1}(U^{\tau-1})\right)\right), \tag{4.6}$$

where $\hat{\sigma}_1 = (\mathcal{I} - h\mathcal{S})^{-1}$ and $\hat{\sigma}_2^{\tau} = (\mathcal{I} - 2h\hat{\mathcal{S}}^{\tau})^{-1}$. Here $\hat{\sigma}_1 : \mathbb{R}^K \to \mathbb{R}^K$ and $\hat{\sigma}_2^{\tau} : \mathbb{R}^K \to \mathbb{R}^K$ are applied pointwise to their input functions.

Algorithm 4.2 The second hybrid splitting method

Data: The initial condition $U(x,0)$.
Result: The computed solution $U(x,T)$.
Set $U^0 = U(x,0)$.
for $\tau = 1, \cdots, M$ **do**
Solve for $U^{\tau-\frac{2}{3}}$:

$$\frac{U^{\tau-\frac{2}{3}} - U^{\tau-1}}{h} = \mathcal{A}^{\tau-1}(U^{\tau-1}) + \hat{\mathcal{S}}^{\tau}(U^{\tau-\frac{2}{3}}) \tag{4.7}$$

Solve for $U^{\tau-\frac{1}{3}}$:

$$\frac{U^{\tau-\frac{1}{3}} - U^{\tau-\frac{2}{3}}}{h} = -G - \alpha\Delta_{\mathcal{W}} U^{\tau-\frac{2}{3}} \tag{4.8}$$

Solve for U^{τ}:

$$\frac{U^{\tau} - U^{\tau-\frac{1}{3}}}{h} \in \mathcal{S}(U^{\tau}) \tag{4.9}$$

end for

4.2. The second splitting algorithm for (3.1). We then consider the sequential splitting of $-G(x) - \alpha\Delta_{\mathcal{W}} U(x,t)$ and $\mathcal{A}(U,t)(x,t) + \hat{\mathcal{S}}(U,t)$, as shown in Algorithm 4.2.

Substep (4.7) is solved as:

$$\begin{aligned} U^{\tau-\frac{2}{3}} &= (\mathcal{I} - h\hat{\mathcal{S}}^{\tau})^{-1}\left(U^{\tau-1} + h\mathcal{A}^{\tau-1}(U^{\tau-1})\right) \\ &= (\mathcal{I} - h\hat{\mathcal{S}}^{\tau})^{-1} \begin{pmatrix} u_1^{\tau-1} + h\sum_{s=1}^{K} \mathbf{a}_{1,s}^{\tau-1} *_{\hat{\mathcal{W}}} u_s^{\tau-1} + hb_1^{\tau-1} \\ u_2^{\tau-1} + h\sum_{s=1}^{K} \mathbf{a}_{2,s}^{\tau-1} *_{\hat{\mathcal{W}}} u_s^{\tau-1} + hb_2^{\tau-1} \\ \vdots \\ u_K^{\tau-1} + h\sum_{s=1}^{K} \mathbf{a}_{K,s}^{\tau-1} *_{\hat{\mathcal{W}}} u_s^{\tau-1} + hb_K^{\tau-1} \end{pmatrix}. \end{aligned}$$

Substep (4.8) is solved explicitly:

$$U^{\tau-\frac{1}{3}} = U^{\tau-\frac{2}{3}} + h(-G - \alpha\Delta_{\mathcal{W}} U^{\tau-\frac{2}{3}}) = \begin{pmatrix} u_1^{\tau-\frac{2}{3}} + h(-g_1 - \alpha\Delta_{\mathcal{W}} u_1^{\tau-\frac{2}{3}}) \\ u_2^{\tau-\frac{2}{3}} + h(-g_2 - \alpha\Delta_{\mathcal{W}} u_2^{\tau-\frac{2}{3}}) \\ \vdots \\ u_K^{\tau-\frac{2}{3}} + h(-g_K - \alpha\Delta_{\mathcal{W}} u_K^{\tau-\frac{2}{3}}) \end{pmatrix}.$$

The last substep (4.9) is solved implicitly:

$$U^{\tau} = (\mathcal{I} - h\mathcal{S})^{-1} U^{\tau-1/3}.$$

The full update from $U^{\tau-1}$ to U^{τ} is given as:

$$U^\tau = \hat{\sigma}_1\left(-hG + (\mathcal{I} - \alpha h\Delta_{\mathcal{W}})\hat{\sigma}_2^\tau\left(U^{\tau-1} + h\mathcal{A}^{\tau-1}(U^{\tau-1})\right)\right). \tag{4.10}$$

4.3. Discretization on finite graphs. After temporal discretization, we discuss the spatial discretization of (3.1) in this subsection. Let $W_N \in \mathbb{R}^{N\times N}$ be an N by N matrix that discretizes $\mathcal{W}$ as: $(W_N)_{ij} = \mathcal{W}(\frac{i-1}{N}, \frac{j-1}{N})$ for $i, j = 1, ..., N$. We discretize the model (3.1) on this finite graph W_N. For each $\tau = 1, \ldots, M$, U^τ and G are discretized as matrices $\mathbf{U}^\tau = (\mathbf{u}_1^\tau, \ldots, \mathbf{u}_K^\tau) \in \mathbb{R}^{N\times K}$ and $\mathbf{G} = (\mathbf{g}_1, \ldots, \mathbf{g}_K) \in \mathbb{R}^{N\times K}$. The identity operator $\mathcal{I}$ is approximated by an identity matrix $\mathbf{I}_N$, the graphon Laplacian $\Delta_{\mathcal{W}}$ is discretized as the graph Laplacian Δ_{W_N} (defined as (A.1) in Appendix A), and the graphon convolution term $\mathcal{A}^{\tau-1}$ is discretized as $A^{\tau-1}$ by replacing $*_{\hat{\mathcal{W}}}$ with the graph convolution $*_{\hat{W}_N}$ (defined as (A.2) in Appendix A), where $\hat{W}_N$ is the discretization of $\hat{\mathcal{W}}(x, y)$ on W_N, and we denote the discretization of $\mathcal{U}_0$ as U_0.

Equations (4.6) and (4.10) are discretized as

$$\mathbf{U}^\tau = \hat{\sigma}_1\left(\frac{1}{2}\mathbf{U}^{\tau-1} + h(-\mathbf{G} + \alpha\Delta_{W_N}\mathbf{U}^{\tau-1}) + \frac{1}{2}\hat{\sigma}_2^\tau\left(\mathbf{U}^{\tau-1} + 2hA^{\tau-1}(\mathbf{U}^{\tau-1})\right)\right), \tag{4.11}$$

and

$$\mathbf{U}^\tau = \hat{\sigma}_1\left(-h\mathbf{G} + (\mathbf{I}_N + \alpha h\Delta_{W_N})\hat{\sigma}_2^\tau\left(\mathbf{U}^{\tau-1} + hA^{\tau-1}(\mathbf{U}^{\tau-1})\right)\right) \tag{4.12}$$

for $\tau = 1, \ldots, M$. Here, the activation functions $\hat{\sigma}_1 : \mathbb{R}^K \to \mathbb{R}^K$ and $\hat{\sigma}_2^\tau : \mathbb{R}^K \to \mathbb{R}^K$ remain the same as in (4.6) and (4.10). The transformation operator $\mathcal{H}$ is discretized as H, and we refer to Appendix D for the definition of H for different tasks.

Finally, the optimal control problem (4.2) becomes

$$\min_\theta \sum_{\hat{i}=1}^{\hat{N}} L(N_\theta(\mathbf{f}_{\hat{i}}), v_{\hat{i}}). \tag{4.13}$$

Here, $\mathbf{f}_{\hat{i}}$ is the discretization of $f_{\hat{i}}$, $N_\theta(\mathbf{f}_{\hat{i}})$ equals $H(\mathbf{U}^M)$, where $\mathbf{U}^M$ is computed by (4.11) or (4.12) iteratively from $\mathbf{U}^0$, and θ denotes the collection of all undetermined variables in $\mathbf{G}$, $\hat{\sigma}_2^\tau$, and $A^{\tau-1}$.

4.4. Connections with message-passing structures of GNNs. The model (3.1) is closely related to many classical message-passing GNNs. In this subsection, we show that some classical GNN layers are operator-splittings of a special case of (3.1) without the region force term G and the diffusion term $\alpha\Delta_{\mathcal{W}}$.

A large class of modern graph neural networks follow similar message-passing structures. A complete graph neural network consists of several message-passing layers and an output layer. The message-passing layers iteratively update nodes' features by combining features from neighboring nodes, and the output layer makes the final prediction according to the outputs of previous layers. The structure of the output layer is identical to H in the previous subsection. We then show that many message-passing layers can be explained using (4.6) or (4.10).

Suppose the matrix $\mathbf{U}^\tau = (\mathbf{u}_1^\tau, \mathbf{u}_2^\tau, \ldots, \mathbf{u}_K^\tau) \in \mathbb{R}^{N\times K}$ contains the hidden state vectors at the τ-th layer. We present the update rule of message-passing layers in some classical GNNs and their connections with (3.1) as follows

- Graph convolutional network (**GCN**) [25]:

The GCN layer (GCNConv) is defined as

$$\mathbf{U}^{\tau} = \sigma\left((\mathbf{I}_N + D_N^{-1/2} W_N D_N^{-1/2})\mathbf{U}^{\tau-1}\mathbf{X}^{\tau-1} + \mathbf{1}_N \otimes (\mathbf{b}^{\tau-1})^{\top}\right) \tag{4.14}$$

where σ is a nonlinear activation function, $\mathbf{X}^{\tau-1} \in \mathbb{R}^{K\times K}$ is the convolution coefficient matrix, $\mathbf{1}_N \in \mathbb{R}^N$ is a vector with all elements equal to 1, and $\mathbf{b}^{\tau-1} \in \mathbb{R}^K$ is the bias vector. The matrix $\mathbf{X}^{\tau-1}$ and vector $\mathbf{b}^{\tau-1}$ are learned during training. Expression (4.14) is exactly a discretization of (4.6) without the region force and the diffusion term on a finite graph W_N. To see this, we first set $G = 0$, $\alpha = 0$, $\hat{\mathcal{S}}^{\tau} = 0$, and absorb the time step $2h$ into the convolution operations $\mathcal{A}$. Then (4.6) is reduced to

$$U^{\tau} = \hat{\sigma}_1\left(U^{\tau-1} + \mathcal{A}^{\tau-1}(U^{\tau-1})\right).$$

We further define each convolution term $\mathbf{a}_{k,s}^{\tau-1}$ in $\mathcal{A}^{\tau-1}$ as

$$\mathbf{a}_{k,s}^{\tau-1} = \begin{cases} (X_{sk}^{\tau-1} - 1, X_{sk}^{\tau-1}), & k = s \\ (X_{sk}^{\tau-1}, X_{sk}^{\tau-1}), & k \neq s \end{cases},$$

where X_{sk} represents the element of $\mathbf{X}$ at s-th row and k-th column, and choose $\hat{\mathcal{W}}$ as $\mathcal{W}(x,y)/\sqrt{d(x)d(y)}$. Then, we have

$$\mathbf{a}_{k,s}^{\tau-1} *_{\hat{\mathcal{W}}} u := \begin{cases} (X_{sk}^{\tau-1} - 1)u + X_{sk}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{\sqrt{d(x)d(y)}}u(y)dy, & k = s, \\ X_{sk}^{\tau-1}u + X_{sk}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{\sqrt{d(x)d(y)}}u(y)dy, & k \neq s, \end{cases}$$

where $X_{sk}^{\tau-1}$ represents the element of $X^{\tau-1}$ at s-th row and k-th column. Then, the term $U^{\tau-1} + \mathcal{A}^{\tau-1}(U^{\tau-1})$ becomes

$$U^{\tau-1} + \begin{pmatrix} \sum_{s=1}^K X_{s1}^{\tau-1}u_s^{\tau-1} \\ \sum_{s=1}^K X_{s2}^{\tau-1}u_s^{\tau-1} \\ \vdots \\ \sum_{s=1}^K X_{sK}^{\tau-1}u_s^{\tau-1} \end{pmatrix} - U^{\tau-1} + \begin{pmatrix} \sum_{s=1}^K X_{s1}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{\sqrt{d(x)d(y)}}u_s^{\tau-1}(y)dy \\ \sum_{s=1}^K X_{s2}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{\sqrt{d(x)d(y)}}u_s^{\tau-1}(y)dy \\ \vdots \\ \sum_{s=1}^K X_{sK}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{\sqrt{d(x)d(y)}}u_s^{\tau-1}(y)dy \end{pmatrix} + \begin{pmatrix} b_1^{\tau-1} \\ b_2^{\tau-1} \\ \vdots \\ b_K^{\tau-1} \end{pmatrix}.$$

If we discretize the system on a finite graph W_N, then $u_k^{\tau-1}$ becomes a vector $\mathbf{u}_k^{\tau-1}$ and the integration $\int_0^1 \frac{\mathcal{W}(x,y)}{\sqrt{d(x)d(y)}}u_s^{\tau-1}(y)dy$ becomes $D_N^{-1/2}W_N D_N^{-1/2}\mathbf{u}_s^{\tau-1}$. Consequently, equation (4.6) is discretized as

$$\mathbf{U}^{\tau} = \hat{\sigma}_1\left(\mathbf{U}^{\tau-1}\mathbf{X}^{\tau-1} + D_N^{-1/2}W_N D_N^{-1/2}\mathbf{U}^{\tau-1}\mathbf{X}^{\tau-1} + \mathbf{1}_N \otimes (\mathbf{b}^{\tau-1})^{\top}\right)$$

where $\mathbf{b}^{\tau-1} = (b_1^{\tau-1}, b_2^{\tau-1}, \ldots, b_K^{\tau-1})^{\top}$. The discretization of (4.6) has exactly the same form as (4.14) with activation $\sigma = \hat{\sigma}_1$.

- **GraphSAGE** [24]:
Consider the SAGE convolution layer (SAGEConv) in [24] with mean aggregation:

$$\mathbf{U}^{\tau} = \sigma\left(\mathbf{U}^{\tau-1}\mathbf{X}^{\tau-1} + D_N^{-1}W_N\mathbf{U}^{\tau-1}\mathbf{Y}^{\tau-1} + \mathbf{1}_N \otimes (\mathbf{b}^{\tau-1})^{\top}\right) \tag{4.15}$$

where $\mathbf{X}^{\tau-1}, \mathbf{Y}^{\tau-1} \in \mathbb{R}^{K\times K}$ are learnable matrices. We set $G(x) = 0$, $\alpha = 0$, $\hat{\mathcal{S}}^\tau = 0$, and absorb the time step into the convolution kernel and the bias term. Equation (4.6) reduces to

$$U^\tau = \hat{\sigma}_1\left(U^{\tau-1} + \mathcal{A}^{\tau-1}(U^{\tau-1})\right).$$

We further define $\mathbf{a}_{k,s}^{\tau-1}$ in $\mathcal{A}^{\tau-1}$ as $\mathbf{a}_{k,s}^{\tau-1} = \begin{cases}(X_{sk}^{\tau-1} - 1, Y_{sk}^{\tau-1}), & k = s\\ (X_{sk}^{\tau-1}, Y_{sk}^{\tau-1}), & k \neq s\end{cases}$, where $X_{sk}^{\tau-1}$ and $Y_{sk}^{\tau-1}$ represent the elements of $\mathbf{X}^{\tau-1}$ and $\mathbf{Y}^{\tau-1}$ at the s-th row and k-th column, and choose $\hat{\mathcal{W}}$ as $\mathcal{W}(x,y)/d(x)$. Then, we have

$$\mathbf{a}_{k,s}^{\tau-1} *_{\hat{\mathcal{W}}} u = \begin{cases}(X_{sk}^{\tau-1} - 1)u + Y_{sk}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{d(x)}u(y)dy & k = s,\\ X_{sk}^{\tau-1}u + Y_{sk}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{d(x)}u(y)dy & k \neq s,\end{cases}$$

and $U^{\tau-1} + \mathcal{A}^{\tau-1}(U^{\tau-1})$ becomes

$$U^{\tau-1} + \begin{pmatrix}\sum_{s=1}^K X_{s1}^{\tau-1}u_s^{\tau-1}(x)\\ \sum_{s=1}^K X_{s2}^{\tau-1}u_s^{\tau-1}(x)\\ \vdots\\ \sum_{s=1}^K X_{sK}^{\tau-1}u_s^{\tau-1}(x)\end{pmatrix} - U^{\tau-1} + \begin{pmatrix}\sum_{s=1}^K Y_{s1}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{d(x)}u_s^{\tau-1}(y)dy\\ \sum_{s=1}^K Y_{s2}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{d(x)}u_s^{\tau-1}(y)dy\\ \vdots\\ \sum_{s=1}^K Y_{sK}^{\tau-1}\int_0^1 \frac{\mathcal{W}(x,y)}{d(x)}u_s^{\tau-1}(y)dy\end{pmatrix} + \begin{pmatrix}b_1^{\tau-1}\\ b_2^{\tau-1}\\ \vdots\\ b_K^{\tau-1}\end{pmatrix}.$$

After further discretizing on W_N, we get the following iterative update

$$\mathbf{U}^\tau = \hat{\sigma}_1\left(\mathbf{U}^{\tau-1}\mathbf{X}^{\tau-1} + D_N^{-1}W_N\mathbf{U}^{\tau-1}\mathbf{Y}^{\tau-1} + \mathbf{1}_N \otimes (\mathbf{b}^{\tau-1})^\top\right),$$

and (4.6) exactly recovers (4.15) with activation $\sigma = \hat{\sigma}_1$.

- Graph isomorphism network (**GIN**) [55]:
The update rule of the GIN layer (GINConv) is

$$\mathbf{U}^\tau = \mathrm{MLP}^\tau((1+\epsilon)\mathbf{U}^{\tau-1} + W_N\mathbf{U}^{\tau-1}) \tag{4.16}$$

where $\mathrm{MLP}^\tau(\cdot)$ is a learnable multilayer perceptron acting on each row of $(1+\epsilon)\mathbf{U}^{\tau-1} + W_N\mathbf{U}^{\tau-1}$, and ϵ is a learnable parameter. Unlike the GCN and GraphSAGE layer, the GIN does not have learnable convolution coefficients and bias. Instead, GIN learns the activation function at each time step.
By setting $G(x) = 0$, $\alpha = 0$, $b_k = 0$, $\mathcal{S} = 0$, and absorbing the time step, (4.10) reduces to

$$U^\tau = \hat{\sigma}_2^\tau\left(U^{\tau-1} + \mathcal{A}^{\tau-1}(U^{\tau-1})\right).$$

By defining the convolution operators in $\mathcal{A}^{\tau-1}$ as

$$\mathbf{a}_{k,s}^{\tau-1} *_{\hat{\mathcal{W}}} u = \begin{cases}\epsilon u + \int_0^1 \frac{\mathcal{W}(x,y)}{d(x)}u(y)dy & k = s\\ 0 & k \neq s,\end{cases} \tag{4.17}$$

and discretizing (4.10) on a finite graph W_N, we get

$$\mathbf{U}^\tau = \hat{\sigma}_2^\tau((1+\epsilon)\mathbf{U}^{\tau-1} + W_N\mathbf{U}^{\tau-1}).$$

Unlike the GCN and GraphSAGE layer, we directly parameterize the activation $\hat{\sigma}_2^\tau$ using an MLP at each time step τ instead of parameterizing $\hat{\mathcal{S}}^\tau$. Then, (4.10) is equivalent to (4.16).

5. The proposed GNN model. Based on the two splitting algorithms discussed in Section 3, we propose two novel GNN models by using the unrolling method [36] for Algorithms 4.1 and 4.2. This procedure involves two steps: discretizing (4.6) and (4.10) on a finite graph W_N, and parameterizing unknown terms, such as $\boldsymbol{G}$, $A^{\tau-1}$, and $\hat{\sigma}_1$ and $\hat{\sigma}_2^\tau$, as learnable modules. The discretization of (4.6) and (4.10) on W_N gives (4.11) and (4.12) respectively. We discuss the parameterization step in the following.

5.1. Parameterization. We need to parameterize the region force term $\mathbf{G}$, the convolution $A^{\tau-1}$, and the activation functions $\hat{\sigma}_1$ and $\hat{\sigma}_2^\tau$. We present a general way of parameterization for each term as follows:

1. The region force terms $\mathbf{G}$:
   We use a learnable GNN module G_{θ^*}, such as the GIN network, to parameterize $\mathbf{G}$, where θ^* denotes the set of all learnable parameters in G_{θ^*}. The input of G_{θ^*} is a graph function $\mathbf{f}$, and the output of G_{θ^*} is a matrix of size $N \times K$, where the k-th column of the output represents the region force $\mathbf{g}_k$. Since $\mathbf{g}_k$ does not depend on τ, G_{θ^*} only needs to be evaluated once.
2. The convolution $A^{\tau-1}$:
   In general, for each $\tau = 1, \dots, M$, the convolution $A^{\tau-1}$ can be parameterized using any graph convolution operations, including those used in GCN, GraphSAGE, and GIN discussed in Section 4.4.
3. The activation $\hat{\sigma}_1$ and $\hat{\sigma}_2^\tau$:
   For the activation $\hat{\sigma}_1 = (\mathcal{I} - h\mathcal{S})^{-1}$, which is time-independent, we can compute it by exact or inexact solvers. The details are discussed in the next subsection. For $\hat{\sigma}_2^\tau = (\mathcal{I} - 2h\hat{\mathcal{S}}^\tau)^{-1}$, instead of solving the inverse problem directly, we parameterize $\hat{\sigma}_2^\tau$ as a learnable MLP module at each time step, like the GIN network, and find its optimal parameters when solving the control problems (4.13).
4. The initialization operator U_0:
   Given an input graph data $\mathbf{f}$, we use a simple GCN convolution layer with a nonlinear activation to parameterize the initialization mapping that lifts the dimension of $\mathbf{f}$ to K and use the output of this convolution layer as $\mathbf{U}^0$.

5.2. On the solution of $\hat{\sigma}_1 = (\mathcal{I} - h\mathcal{S})^{-1}$. The nonlinear term $\hat{\sigma}_1$ serves as the activation function in our GNN. Here, we discuss how to compute $\hat{\sigma}_1$ for different choices of $\mathcal{S}$. Since $\mathcal{S}(U)$ is applied to each $U(x) \in \mathbb{R}^K$ pointwise, it is sufficient to show how to compute $\hat{\sigma}_1(\mathbf{y})$ for a given vector $\mathbf{y} \in \mathbb{R}^K$.

In (2.3), the nonlinear term is defined as $\mathcal{S}(U) = -\beta - \beta\log(U) - \partial I_{\widetilde{\Delta}^K}(U)$, and $\hat{\sigma}_1(\mathbf{y})$ aims to solve for a vector $\mathbf{x}$ such that

$$\frac{\mathbf{x}-\mathbf{y}}{h} \in \mathcal{S}(\mathbf{x}) \Leftrightarrow \frac{\mathbf{x}-\mathbf{y}}{h} + \beta + \beta\log(\mathbf{x}) + \partial I_{\widetilde{\delta}^K}(\mathbf{x}) \ni \mathbf{0}, \tag{5.1}$$

where $I_{\widetilde{\delta}^K}$ is the indicator function of the set

$$\widetilde{\delta}^K := \left\{ \mathbf{z} \in [0,1]^K \,\middle|\, \sum_{k=1}^K \mathbf{z}_k = 1 \right\}.$$

To solve (5.1), we consider a similar two-substep splitting technique as [49]:

$$\begin{cases} \text{Substep 1. solve for } \mathbf{x}' : \frac{\mathbf{x}'-\mathbf{y}}{h} + \beta + \beta\log(\mathbf{x}') = 0, \\ \text{Substep 2. solve for } \mathbf{x} : \frac{\mathbf{x}-\mathbf{x}'}{h} + \partial I_{\widetilde{\delta}^K}(\mathbf{x}) \ni 0. \end{cases} \tag{5.2}$$

The first substep can be solved by a fixed-point iteration:

$$\frac{(\mathbf{x}')^\gamma - \mathbf{y}}{h\beta} + 1 + \log((\mathbf{x}')^{\gamma+1}) = 0 \Rightarrow (\mathbf{x}')^{\gamma+1} = \exp\left(\frac{\mathbf{y} - (\mathbf{x}')^\gamma}{h\beta} - 1\right),$$

where $\gamma = 0, 1, \ldots$ denotes the fixed-point iteration step. Since $\mathbf{x}$ gets updated from iteration to iteration when we solve (2.3), we only need to find an approximate solution for this substep for efficiency considerations. We follow [29] to use a few steps of this fixed-point iteration. The $(\mathbf{x}')^\gamma$ after the last fixed-point iteration step is used as the final solution $\mathbf{x}'$. The second substep can be solved by a simple projection of the vector $\mathbf{x}'$ to the set $\widetilde{\delta}^K$. Since all elements of $\mathbf{x}'$ are positive, we can compute this projection by

$$\mathbf{x} = \mathbf{x}' / \sum_{k=1}^{K} (\mathbf{x}')_k.$$

Note that if we only do 1 fixed-point iteration for the first substep and choose $(\mathbf{x}')^0 = 0$, we have

$$\hat{\sigma}_1(\mathbf{y}) = \exp(\mathbf{y}/(h\beta) - 1) / \sum_{k=1}^{K} \exp(\mathbf{y}_k/(h\beta) - 1) = \text{Softmax}(\mathbf{y}/(h\beta)), \tag{5.3}$$

which exactly recovers the softmax function when $h\beta = 1$.

In a more general setting, when the problem is not a classification problem, it is not necessary to require that $U \in \widetilde{\Delta}^K$ in (2.3). Instead, we can replace the regularization $\sum_{k=1}^{K} \langle u_k, \log(u_k) \rangle_{\mathcal{W}}$ by $\sum_{k=1}^{K} (\langle u_k, \log(u_k) \rangle_{\mathcal{W}} + \langle 1 - u_k, \log(1 - u_k) \rangle_{\mathcal{W}})$ and require $U \in [0, 1]^K$ only. In this case, the resulting nonlinear term $\mathcal{S}(U)$ in (2.3) becomes $-\beta \log(U/(1 - U))$, and $\hat{\sigma}_1(\mathbf{y})$ is equivalent to solving

$$\frac{\mathbf{x} - \mathbf{y}}{h} = -\beta \log(\mathbf{x}/(1 - \mathbf{x})).$$

We can also use a fixed-point iteration like [48] to solve it

$$\frac{(\mathbf{x})^\gamma - \mathbf{y}}{h} = -\beta \log\left((\mathbf{x})^{\gamma+1}/(1 - (\mathbf{x})^{\gamma+1})\right).$$

Similarly, if only 1 fixed-point iteration is used and we set $(\mathbf{x})^0 = 0$, we have

$$\hat{\sigma}_1(\mathbf{y}) = \text{Sigmoid}(\mathbf{y}/(h\beta)) \tag{5.4}$$

which is exactly a sigmoid function when $h\beta = 1$.

Besides, instead of directly parameterizing $\mathcal{S}$, we can also parameterize $\hat{\sigma}_1$ with some common activation function, like the Tanh function and ReLU function.

5.3. GraphPottsNet. After parameterization at each time step, we obtain two novel GNN structures based on Algorithms 4.1 and 4.2. Since Algorithms 4.1 and 4.2 solve a control problem originally derived from the Potts model, we call our new GNNs GraphPottsNet (GPN). According to the treatment when dealing with the operator groups (sequential or parallel), we denote the unrolled networks from Algorithms 4.1 and 4.2 as GPN(par) and GPN(seq), respectively. The graphical illustrations of the two GPN structures are shown in Figure 1. In both GNN models, the first layer is a simple graph convolution layer [25], which aims to change the data dimension, and the last layer is a task-dependent transformation layer, which aggregates all information and makes

final predictions. The two GPNs are derived from hybrid splitting algorithms with the same order of accuracy, as shown in Appendix F, so we would expect them to have similar performance in practice. Here, we present both algorithms to show the flexibility and generality of our framework. In the later numerical experiments, we will show that the two GPNs indeed have similar performance.

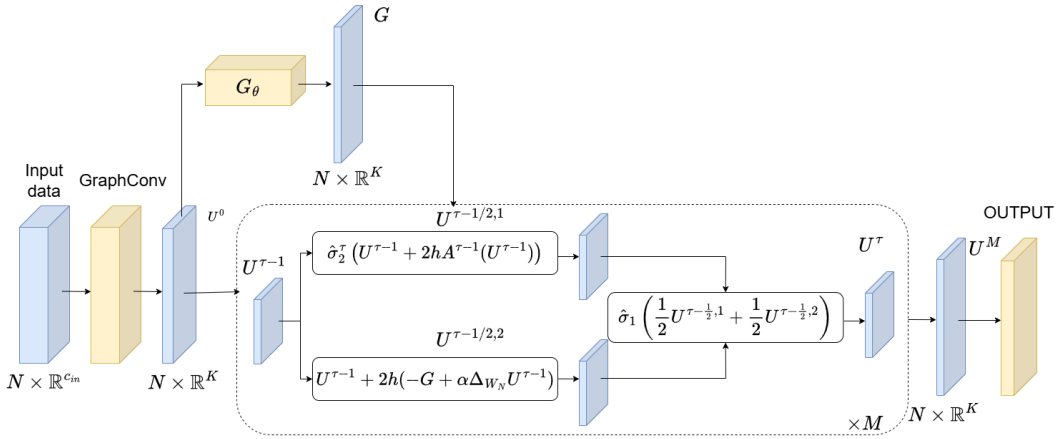
Compared to existing GNN models discussed in Section 4.4, our proposed models are derived from the gradient flow of the graph Potts model, which provides a solid mathematical foundation. Although both classical GNN models and our proposed models fall within our framework discussed in Section 3, ours are distinguished by an extra explicit region force term and diffusion term. The learnable region force term $\mathbf{G}$ can incorporate data-dependent information and drive the iterative scheme to obtain a desired solution. The diffusion term, which is derived from the boundary regularization in the original Potts model, can prevent the GNNs from overfitting the training data and provide some regularization effect. Therefore, our proposed model would maintain high expressiveness and robustness simultaneously.

In this work, the continuous graphon viewpoint serves primarily as a theoretical and organizational framework for deriving the proposed network architectures and interpreting existing GNN models. More specifically, the graphon formulation provides a continuous perspective that helps motivate the control problem, the operator-splitting scheme design, and the connection with existing message-passing GNN layers. The actual GNNs used in the experiments are finite-graph discretizations of these constructions, and they are trained and evaluated directly on finite datasets. At the same time, we do not mean to suggest that this continuous interpretation is always meaningful in practice for finite graph datasets. Even though it is possible to mathematically write a finite graph together with its node features as a discretization of a continuous graphon and a graphon signal, for many real data like the protein structure graphs, this continuous counterpart does not have a direct physical or practical interpretation.

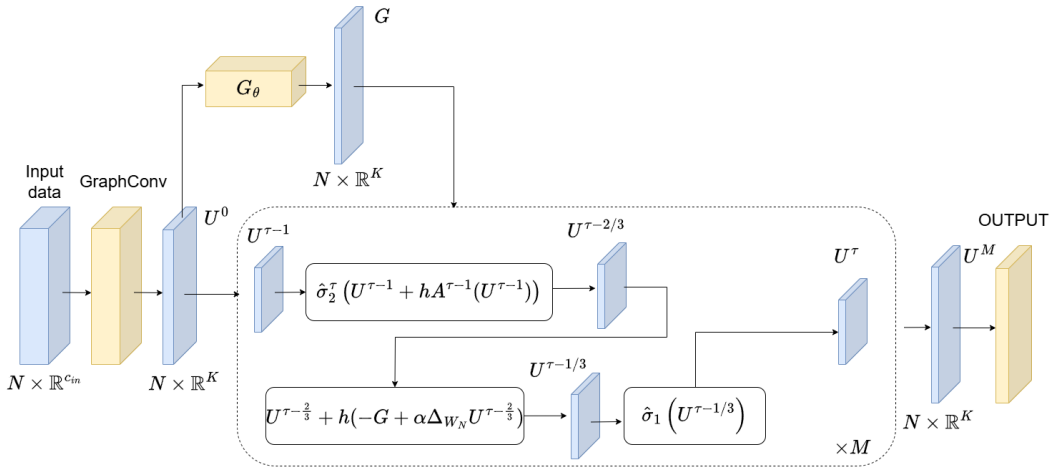
6. Numerical experiments. In this section, we demonstrate the effectiveness of the proposed GPN models via several tasks. For comparison, we choose GCN [25], GraphSAGE (GSG) [24], Graph Transformer (GT) [16], and GIN [55] as baseline models. Even though the GT model is not directly related to our framework, it is included as a strong baseline for comparison. All baseline models are implemented using their standard DGL implementations [53]. Implementation details, hyper-parameter choices, and computational costs are reported in Appendix G.

6.1. Graph regression. We first compare our models with others on the graph regression tasks on the QM9Dataset [39]. The QM9Dataset is a widely used benchmark dataset in computational chemistry and molecular machine learning. It contains 133,885 molecular graphs and provides 12 different regression targets. In this work, we choose 4 energy-related regression targets, including internal energy at 0K, internal energy at 298.15K, enthalpy at 298.15K, and free energy at 298.15K, to test our models. The performances of different models are measured by the relative Mean Absolute Error (MAE). For each case, we conduct 5 trials to evaluate each model. In each trial, we randomly split the whole dataset into training (60%), validation (20%), and testing (20%). The mean and standard deviation of the testing accuracy of all models are presented in Table 1. In this set of experiments, our proposed GPNs consistently achieve smaller prediction error than other GNN models, while the GIN model also obtains accuracy comparable to ours (the difference is within one standard deviation). Our GPNs and the GIN model are significantly better than the other models in terms of prediction error and standard deviation.

6.2. Node classification. The second task we consider is the node classification task. We choose several popular citation network datasets [44] listed in Table 2. Each dataset consists of a



(a) GPN(par)



(b) GPN(seq)

Fig. 1: GraphPottsNet structures based on Algorithm 4.1 (GPN(par)) and Algorithm 4.2 (GPN(seq)).

single large graph, where each node in the graph represents a paper, and edges represent citation relationships between papers. The node features represent the frequency of words from a given dictionary appearing in the paper, and the labels represent the research field of each paper. Given a portion of nodes with known labels, the task is to predict the research field of other papers based on the graph structure and node features. We randomly split all nodes in the graph into training (10%), validation (30%), and test (60%) sets, and report the average accuracy and standard

	internal energy at 0K	internal energy at 298.15K	enthalpy at 298.15K	free energy at 298.15K
GCN	0.048 (±0.0015)	0.050 (±0.0025)	0.045 (±0.0007)	0.049 (±0.0027)
GraphSAGE	0.046 (±0.0058)	0.041 (±0.0042)	0.040 (±0.0026)	0.044 (±0.0067)
GT	0.037 (±0.0015)	0.036 (±0.0037)	0.036 (±0.0015)	0.037 (±0.0026)
GIN	**0.014** (±**0.0003**)	**0.014** (±**0.0002**)	**0.015** (±**0.0003**)	**0.014** (±**0.0003**)
GPN(par)	**0.013*** (±**0.0018**)	**0.013*** (±**0.0002**)	**0.014*** (±**0.0001**)	**0.013*** (±**0.0001**)
GPN(seq)	**0.013*** (±**0.0013**)	**0.013*** (±**0.0001**)	**0.014*** (±**0.0002**)	**0.013*** (±**0.0005**)

Table 1: Relative MAE and standard deviation of different models tested on different regression targets from the QM9Dataset. In each column, the smallest MAE is marked in bold with an asterisk (*). Other bold entries indicate mean accuracies whose difference from the smallest MAE is less than one standard deviation of the best result.

deviation of different models on the test set over 5 trials.

To evaluate the robustness of different models, we also add Gaussian noise with different levels to the node features and test the performance of different models under noisy conditions. Specifically, for a given node feature vector, we add random Gaussian noise with standard deviation σ to each element independently. Then, we first clip the resulting noisy feature vector to be non-negative and further normalize it by dividing the sum of all elements to make its l_1 norm equal to 1. We test the performance of different models under three different noise levels, including $\sigma = 0.00$ (indicating no noise), $\sigma = 0.01$, and $\sigma = 0.05$. All numerical results are presented in Table 3.

When there is no noise, our proposed GPNs always achieve the highest mean accuracy, while the other models may also obtain comparable performance. When noise is added, the performance of all models degrades, but our proposed GPNs show high robustness and achieve significantly higher classification accuracy than other models in most cases. The performance gap between our proposed GPNs and other models becomes larger when the noise level increases, which indicates that our proposed GPNs are more robust to noise than other models.

Dataset	Number of nodes	Number of edges	Node feature dimension	Number of classes
CORA	2708	10556	1433	7
CITESEER	3327	9228	3703	6
PUBMED	19717	88651	500	3

Table 2: Descriptions of three node classification datasets.

Dataset	σ	GCN	GraphSAGE	GT	GIN	GPN(par)	GPN(seq)
CORA	0.00	**81.3%(0.6%)**	81.1%(0.9%)	76.0%(1.2%)	77.9%(1.0%)	**81.8%(0.7%)**	**82.3%(1.0%)***
	0.01	78.4%(0.9%)	77.7%(0.4%)	65.5%(2.8%)	73.6%(1.6%)	**81.0%(0.8%)**	**81.2%(1.6%)***
	0.05	64.1%(1.8%)	60.1%(0.7%)	54.9%(4.3%)	63.7%(2.2%)	**73.3%(1.1%)***	**72.4%(1.8%)**
CITESEER	0.00	**68.7%(1.1%)**	**68.6%(0.9%)**	59.5%(2.5%)	64.7%(1.0%)	**68.9%(1.3%)**	**69.3%(0.9%)***
	0.01	56.5%(1.8%)	55.9%(0.7%)	46.7%(2.2%)	49.7%(3.6%)	**63.6%(1.9%)***	**63.5%(0.9%)**
	0.05	41.3%(1.8%)	39.4%(0.6%)	41.2%(1.8%)	43.4%(1.5%)	**50.2%(2.1%)***	**49.1%(2.6%)**
PUBMED	0.00	**82.9%(0.3%)**	81.4%(0.4%)	80.6%(0.3%)	**82.8%(0.2%)**	**83.2%(0.4%)***	**83.2%(0.4%)***
	0.01	82.4%(0.4%)	80.9%(0.4%)	78.0%(1.2%)	81.8%(0.7%)	**82.9%(0.4%)**	**83.0%(0.3%)***
	0.05	73.7%(0.4%)	67.6%(0.5%)	43.6%(0.8%)	73.1%(3.0%)	**77.6%(0.2%)***	**77.3%(0.6%)**

Table 3: Classification accuracy of different models on the citation networks under different Gaussian noise levels. In each row, the highest mean accuracy is marked in bold with an asterisk (*). Other bold entries indicate mean accuracies whose difference from the highest mean accuracy is less than one standard deviation of the best result.

Dataset	σ	GPN(par) $\alpha = 0$	GPN(par) $\alpha = 1$	GPN(par) $\alpha = 2$	GPN(seq) $\alpha = 0$	GPN(seq) $\alpha = 1$	GPN(seq) $\alpha = 2$
CORA	0.00	81.4%(1.2%)	81.8%(0.7%)	**82.9%(0.3%)***	**81.4%(1.8%)**	**82.3%(1.0%)***	**82.1%(0.9%)**
	0.01	**80.8%(1.2%)**	**81.0%(0.8%)***	**80.6%(0.8%)**	78.9%(1.0%)	**81.2%(1.6%)**	**81.5%(1.4%)***
	0.05	65.9%(3.4%)	**73.3%(1.1%)**	**73.8%(0.7%)***	63.3%(0.5%)	**72.4%(1.8%)**	**72.7%(2.0%)***
CITESEER	0.00	**68.8%(1.3%)**	**68.9%(1.3%)**	**69.2%(1.5%)***	**68.1%(0.6%)**	**69.3%(0.9%)***	**68.8%(1.1%)**
	0.01	60.4%(1.7%)	**63.6%(1.9%)***	**63.1%(1.5%)**	57.0%(2.1%)	**63.5%(0.9%)***	**62.4%(1.3%)**
	0.05	39.1%(2.5%)	**50.2%(2.1%)***	46.2%(4.1%)	41.4%(1.2%)	49.1%(2.6%)	**51.4%(2.1%)***
PUBMED	0.00	**83.2%(0.3%)***	**83.2%(0.4%)***	**83.0%(0.4%)**	**83.2%(0.3%)***	**83.2%(0.4%)***	**83.2%(0.3%)***
	0.01	**82.9%(0.4%)**	**82.9%(0.4%)**	**83.1%(0.2%)***	**82.9%(0.5%)**	**83.0%(0.3%)**	**83.1%(0.2%)***
	0.05	75.8%(1.1%)	77.6%(0.2%)	**79.2%(0.5%)***	73.3%(0.6%)	**77.3%(0.6%)**	**77.7%(0.9%)***

Table 4: Classification accuracy of GPNs with different α on the citation networks under different Gaussian noise levels. In each half row (the first three or the last three entries), the highest mean accuracy is marked in bold with an asterisk (*). Other bold entries indicate mean accuracies whose difference from the highest mean accuracy is less than one standard deviation of the best result.

Besides, we also test the performance of our proposed GPNs with different α values ($\alpha = 0, 1, 2$) under different noise levels. The results are shown in Table 4. We can see that when there is no noise, the mean accuracies are comparable for different α values. When noise is added, $\alpha = 1$ and $\alpha = 2$ always achieve significantly better performance than $\alpha = 0$, which indicates that the diffusion term makes a non-trivial improvement to the robustness of our proposed GPNs. Besides, in most cases, the mean accuracy does not further improve when we increase α from 1 to 2. It indicates that $\alpha = 1$ is generally good enough for this kind of task, *i.e.*, the GPNs with $\alpha = 1$ can obtain competitive performance as the state-of-the-art GNN models on clean data and also maintain significantly higher robustness under noisy conditions.

6.3. Graph classification. Graph classification is a fundamental task in machine learning whose objective is to predict the label or category of an entire graph based on its structure and features. The features of graphs are usually represented by a high-dimensional graph signal. We

test our proposed methods on four graph classification datasets from the TUDataset [37], including binary classification and multiclass classification. They are described in Table 5.

Dataset	Number of graphs	Average nodes	Node feature dimension	Number of classes
MUTAG	188	17.9	7	2
NCI1	4110	41.8	37	2
IMDBBINARY	1000	13.0	1	2
ENZYMES	600	32.6	18	6

Table 5: Descriptions of four graph classification datasets from the TUDataset [37].

For each dataset, we conduct 5 trials to evaluate each model. In each trial, we randomly split the whole dataset into training (60%), validation (20%), and testing (20%). The mean and standard deviation of the testing accuracy of all models are presented in Table 6. In the numerical results, the GPN(par) achieves the highest accuracy in the MUTAG and ENZYMES datasets, while the GPN(seq) achieves the highest accuracy in the NCI1 and IMDBBINARY datasets. The GIN network also obtains a high accuracy comparable to our proposed GPNs (the difference is within one standard deviation). We observe that the accuracy of GPNs and GIN is significantly higher than that of the other three methods. Most methods maintain a reasonably small standard deviation.

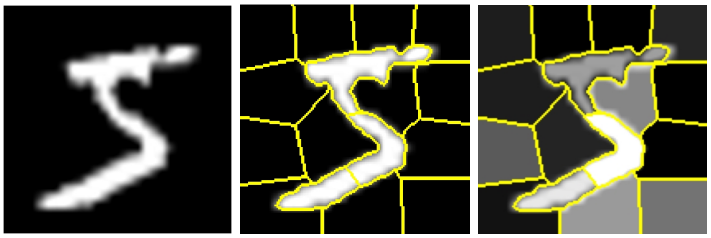
Superpixel graph classification with random node perturbations. To evaluate the robustness of our proposed GPNs against random noise, we consider classification problems on superpixel graphs with random node feature perturbations. We choose the MNIST superpixel dataset [17], which is a 10-class classification dataset. The MNIST superpixel dataset is a collection of hand-written digit images represented using graph structures. Here, each node corresponds to a superpixel, and edges represent relationships, e.g., adjacency or similarity, between superpixels. A superpixel is a group of connected pixels in an image that share similar characteristics, such as color, texture, or intensity. The feature of the superpixel is the average intensity values of the corresponding groups of pixels. Using this graph representation can reduce the complexity of an image while preserving the essential structure and boundaries of objects of interest. The superpixel graph can be constructed using some common clustering algorithms.

Images are very likely to be perturbed by noise, so we also add some random Gaussian noise to the node features of superpixel graphs. A graphical illustration is shown in Figure 2. We first present a sample image (subfigure (a)) from the MNIST dataset, and this image is partitioned into 16 superpixels (subfigure (b)) based on their intensities. Each superpixel will be regarded as a node in the corresponding superpixel graph, and the node feature is the feature vector of each superpixel. We remark that we use a high-resolution version of the MNIST image to present in Figure 2 for a better visual quality, while the superpixel graph dataset is constructed from the original MNIST dataset [14] where each image is of size 28 by 28. In this experiment, we add random noise to perturb the node features of each superpixel node (subfigure (c)).

We first consider the case when the same level of Gaussian noise (noise with the same standard deviation σ) is added to the training, validation, and testing sets. The prediction results are shown in Table 7. Under the clean setting ($\sigma = 0$), the GT model achieves the highest accuracy while the GIN and our GPNs also achieve competitive results. Under the noisy setting ($\sigma > 0$), our

Model	MUTAG	NCI1	IMDBBINARY	ENZYMES
GCN	73.5%(±5.8%)	64.0%(±1.5%)	64.8%(±3.2%)	41.5%(±7.5%)
GraphSAGE	74.6%(±9.3%)	68.9%(±1.6%)	49.6%(±1.3%)	43.2%(±5.5%)
GT	78.2%(±4.2%)	69.5%(±2.3%)	63.1%(±3.5%)	46.3%(±9.4%)
GIN	**83.6**%(±**3.7**%)	**79.2**%(±**1.6**%)	**71.2**%(±**2.9**%)	**55.3**%(±**2.7**%)
GPN(par)	**86.2**%(±**4.5**%)*	**79.8**%(±**1.6**%)	**71.9**%(±**3.6**%)	**57.5**%(±**5.9**%)*
GPN(seq)	**84.6**%(±**3.8**%)	**80.0**%(±**1.3**%)*	**72.5**%(±**1.7**%)*	**54.8**%(±**3.8**%)

Table 6: Classification accuracy and standard deviation of different models on graph classification dataset. In each column, the highest mean accuracy is marked in bold with an asterisk (*). Other bold entries indicate mean accuracies whose difference from the highest mean accuracy is less than one standard deviation of the best result.



(a) Original image (b) Superpixel representation (c) Noisy superpixel representation

Fig. 2: Illustration of superpixel graphs with Gaussian noise. Subfigure (a) is a clean image in the original MNIST dataset. Subfigure (b) is a superpixel representation of this image and each patch represents a superpixel. Subfigure (c) is obtained by adding random Gaussian noise to each superpixel in (b).

proposed GPNs achieve higher classification accuracy than other GNN models and outperform the other models by a large margin, indicating the robustness of our GPNs against node perturbations. We also test the cases when the training set and the testing set are perturbed by different levels of noise. The results are shown in Table 8. Here, σ_{train} is the standard deviation of the noise added to the training set, and σ_{test} is the noise added to the testing set. In most of the testing scenarios, our GPNs can achieve significantly higher classification accuracy, demonstrating the superior robustness of our GPNs.

Effects of α and the G_θ network. Here, we study the effects of different choices of α and G_θ using the MNIST superpixel dataset.

In the control problem (3.1), parameter α controls the strength of regularization. Table 9 shows that when $\sigma = 0$, changing α only causes small differences. In contrast, under heavy noise, the effect of regularization is significant: larger α (especially $\alpha = 10$ or 20) consistently gives much higher test accuracy than $\alpha = 0$. For example, for GPN(par) trained when $\sigma_{train} = \sigma_{test} = 1$, increasing α from 0 to 10 improves accuracy from 30.4% to 38.0%. These results indicate that the

models	test accuracy		
	$\sigma = 0$	$\sigma = 0.5$	$\sigma = 1$
GCN	82.8%	21.8%	13.3%
GraphSAGE	87.8%	22.6%	15.7%
GT	**94.0**%*	22.0%	14.4%
GIN	91.9%	46.9%	27.6%
GPN(par)	91.5%	**57.9**%*	**38.0**%*
GPN(seq)	92.0%	57.5%	35.5%

Table 7: Classification accuracy on the MNIST superpixel graph dataset with different levels of Gaussian noise added to node features (training and testing sets are perturbed by the Gaussian noise with standard deviation σ). The highest mean accuracy in each column is marked in bold with an asterisk (*).

σ_{train}	models	test accuracy		
		$\sigma_{test} = 0.$	$\sigma_{test} = 0.5$	$\sigma_{test} = 1$
0.	GCN	82.8%	10.3%	9.3%
	GSG	87.9%	11.6%	10.4%
	GT	**94.0**%*	12.2%	9.6%
	GIN	92.0%	10.2%	9.2%
	GPN(par)	91.5%	**12.3**%*	**10.9**%*
	GPN(seq)	92.0%	11.2%	10.4%
0.5	GCN	34.8%	21.8%	12.2%
	GSG	26.0%	22.7%	12.7%
	GT	18.8%	22.1%	11.4%
	GIN	58.5%	46.9%	18.2%
	GPN(par)	**68.1**%*	**57.9**%*	**32.1**%*
	GPN(seq)	67.9%	57.5%	27.8%
1	GCN	11.4%	12.3%	13.3%
	GSG	13.7%	15.8%	15.6%
	GT	12.7%	13.7%	14.3%
	GIN	32.9%	29.4%	27.6%
	GPN(par)	**42.2**%*	**41.0**%*	**38.0**%*
	GPN(seq)	38.4%	38.0%	35.5%

Table 8: Classification accuracy on the MNIST superpixel graph dataset with different levels of Gaussian noise added to node features (training and testing sets are perturbed by different noise levels). σ_{train} and σ_{test} represent the standard deviation of noise added to the training and testing sets respectively. The highest mean accuracy in each column of each block is marked in bold with an asterisk (*).

diffusion regularization is crucial for robustness in heavily noisy settings.

σ_{train}	models	α	test accuracy		
			$\sigma_{test} = 0.$	$\sigma_{test} = 0.5$	$\sigma_{test} = 1$
0.	GPN(par)	0.	90.3%	11.4%	10.5%
	GPN(par)	10.	91.5%	**12.3**%*	**10.9**%*
	GPN(par)	20.	92.1%	10.8%	10.5%
	GPN(seq)	0.	91.0%	10.7%	9.8%
	GPN(seq)	10.	92.0%	11.2%	10.4%
	GPN(seq)	20.	**93.4**%*	12.2%	10.1%
0.5	GPN(par)	0.	**69.5%***	57.4%	26.2%
	GPN(par)	10.	68.1%	**57.9%***	32.1%
	GPN(par)	20.	66.8%	57.6%	**32.2%***
	GPN(seq)	0.	64.2%	53.0%	23.7%
	GPN(seq)	10.	68.0%	57.5%	27.8%
	GPN(seq)	20.	62.6%	56.5%	27.4%
1	GPN(par)	0.	34.1%	33.8%	30.4%
	GPN(par)	10.	**42.2%***	**41.0%***	**38.0%***
	GPN(par)	20.	40.5%	40.4%	37.3%
	GPN(seq)	0.	32.3%	32.0%	30.3%
	GPN(seq)	10.	38.4%	38.1%	35.5%
	GPN(seq)	20.	37.3%	36.8%	34.7%

Table 9: Classification accuracy of the proposed models when we choose different values for α. Here, we add Gaussian noise with the same standard deviation σ to both the training set and the testing set. In each σ_{train} block and each test-noise column, the highest accuracy is marked in bold with an asterisk (*).

We then investigate the effects of different parameterizations of the region force term G_θ. In previous experiments (Table 7 and Table 8), we parameterize G_θ using the GIN network structure. Here, we change the structure of G_θ to GCN, GSG, and GT respectively, and evaluate the proposed GPNs on the MNIST superpixel dataset with different levels of noise. The results are shown in Table 10, showing that the proposed GPNs significantly improve the robustness regardless of the structures of G_θ.

7. Discussion and conclusion. We proposed a framework to construct GNN models from control problems. We showed that many classical GNN models are operator-splitting methods of certain graphon control problems. Based on these findings, we proposed GraphPottsNets based on the gradient flow of the graphon Potts model. The proposed model can achieve better results than existing methods in various tasks. Moreover, our models are more robust to graph node feature perturbations than benchmark models and can achieve good prediction accuracy when the input graph is disturbed by intensive noise.

In our numerical experiments, we observe that the two proposed GPN models, i.e., GPN(par) and GPN(seq), have similar performance in most cases. The main difference is structural rather than in predictive accuracy. In Appendix F, we show that both splitting algorithms have the same

models	G_θ	test accuracy		
		$\sigma = 0.25$	$\sigma = 0.5$	$\sigma = 1.0$
GPN(par)	GCN	80.4%	54.8%	34.2%
	GSG	78.7%	52.2%	32.1%
	GT	80.1%	55.9%	41.7%
	GIN	80.1%	57.7%	35.7%
GPN(seq)	GCN	79.4%	55.0%	33.2%
	GSG	77.2%	51.3%	31.3%
	GT	80.6%	54.9%	39.5%
	GIN	80.3%	55.4%	35.1%

Table 10: Classification accuracy of the proposed models when we use different structures to parameterize G_θ. Here, we add Gaussian noise with the same standard deviation to both the training set and the testing set.

first-order accuracy. Therefore, it is natural to expect similar performance in practice for these two methods. The practical difference is mainly that the parallel splitting allows certain operations to be carried out simultaneously, which may bring slightly better efficiency, while the sequential splitting applies the same ingredients step by step. Since the two variants are based on the same underlying framework and differ mainly in how the updates are organized, we do not expect a significant performance gap between them. We present both variants to show the flexibility of the framework rather than to claim that one is universally better than the other. In practice, the parallel splitting can be a reasonable default because of its slightly better efficiency, while the sequential splitting is also a valid choice.

In this work, our proposed GPN models demonstrate robustness to node feature perturbations. However, enhancing robustness against graph structure perturbations remains challenging. In future research, we will explore extending our framework to construct novel GNN models based on different control problems with different regularization terms to improve the robustness against graph structure perturbations.

Code Availability. The code for the proposed GPN models can be found in https://github.com/LingfengLi-MATH/GraphPottsNet

Acknowledgment. The work of Lingfeng Li was partially supported by HKRGC GRF Grant LU13300125. The work of Hao Liu was partially supported by HKRGC GRF 12301925 and 12302926, and Guangdong and Hong Kong Universities "1+1+1" Joint Research Collaboration Scheme 2025A0505000007. The work of Xue-Cheng Tai was partially supported by NORCE Kompetanseoppbygging program. The work of Raymond Chan was partially supported by HKRGC grants LU13300125 and LU11309922, ITF grant MHP/054/22, and LUBGR105824.

REFERENCES

[1] J. Ashkin and E. Teller. Statistics of two-dimensional lattices with four components. *Physical Review*, 64(5-6):178, 1943.

[2] M. Bacák, R. Bergmann, G. Steidl, and A. Weinmann. A second order nonsmooth variational model for restoring manifold-valued images. *SIAM Journal on Scientific Computing*, 38(1):A567–A597, 2016.
[3] A. L. Bertozzi and A. Flenner. Diffuse interface models on graphs for classification of high dimensional data. *Multiscale Modeling & Simulation*, 10(3):1090–1118, 2012.
[4] A. L. Bertozzi and A. Flenner. Diffuse interface models on graphs for classification of high dimensional data. *Siam Review*, 58(2):293–328, 2016.
[5] A. L. Bertozzi, X. Luo, A. M. Stuart, and K. C. Zygalakis. Uncertainty quantification in graph-based classification of high dimensional data. *SIAM/ASA Journal on Uncertainty Quantification*, 6(2):568–595, 2018.
[6] C. Borgs, J. T. Chayes, L. Lovász, V. T. Sós, and K. Vesztergombi. Convergent sequences of dense graphs i: Subgraph frequencies, metric properties and testing. *Advances in Mathematics*, 219(6):1801–1851, 2008.
[7] C. Borgs, J. T. Chayes, L. Lovász, V. T. Sós, and K. Vesztergombi. Convergent sequences of dense graphs ii. multiway cuts and statistical physics. *Annals of Mathematics*, pages 151–219, 2012.
[8] A. Braides, P. Cermelli, and S. Dovetta. γ-limit of the cut functional on dense graph sequences. *ESAIM: Control, Optimisation and Calculus of Variations*, 26:26, 2020.
[9] T. Chan and L. Vese. An active contour model without edges. In *International Conference on Scale-space Theories in Computer Vision*, pages 141–151. Springer, 1999.
[10] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.
[11] S. Chen, J. He, and X.-C. Tai. Neural flow operators can approximate any operator: Abstract frameworks and universal approximations, 2026.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.
[13] M. D. Cranmer, R. Xu, P. Battaglia, and S. Ho. Learning symbolic physics with graph networks. *arXiv preprint arXiv:1909.05862*, 2019.
[14] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
[15] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in Neural Information Processing Systems*, 28, 2015.
[16] V. P. Dwivedi and X. Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.
[17] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.
[18] L. C. Evans. *Partial differential equations*, volume 19. American mathematical society, 2022.
[19] C. Garcia-Cardona, E. Merkurjev, A. L. Bertozzi, A. Flenner, and A. G. Percus. Multiclass data segmentation using diffuse interface methods on graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1600–1613, 2014.
[20] S. Geisler, T. Schmidt, H. Şirin, D. Zügner, A. Bojchevski, and S. Günnemann. Robustness of graph neural networks at scale. *Advances in Neural Information Processing Systems*, 34:7637–7649, 2021.
[21] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.
[22] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 399–406, 2010.
[23] A. Griffa, B. Ricaud, K. Benzi, X. Bresson, A. Daducci, P. Vandergheynst, J.-P. Thiran, and P. Hagmann. Transient networks of spatio-temporal connectivity map communication pathways in brain functional systems. *NeuroImage*, 155:490–502, 2017.
[24] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.
[25] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.
[26] E. Kobler, T. Klatzer, K. Hammernik, and T. Pock. Variational networks: connecting variational methods and deep learning. In *Pattern Recognition: 39th German Conference, GCPR 2017, Basel, Switzerland, September 12–15, 2017, Proceedings 39*, pages 281–293. Springer, 2017.
[27] Y. Lan, Z. Li, J. Sun, and Y. Xiang. Dosnet as a non-black-box pde solver: When deep learning meets operator splitting. *Journal of Computational Physics*, 491:112343, 2023.
[28] R. Levie. A graphon-signal analysis of graph neural networks. *Advances in Neural Information Processing Systems*, 36:64482–64525, 2023.
[29] H. Liu, J. Liu, R. H. Chan, and X.-C. Tai. Double-well net for image segmentation. *Multiscale Modeling &*

Simulation, 22(4):1449–1477, 2024.
[30] H. Liu, X.-C. Tai, and R. Chan. Connections between operator-splitting methods and deep neural networks with applications in image segmentation. *Annals of Applied Mathematics*, 39(4):406–428, 2023.
[31] J. Liu, X. Wang, and X.-C. Tai. Deep convolutional neural networks with spatial regularization, volume and star-shape priors for image segmentation. *Journal of Mathematical Imaging and Vision*, 64(6):625–645, 2022.
[32] L. Lovász. *Large networks and graph limits*, volume 60. American Mathematical Society, 2012.
[33] J. Lu, Z. Shen, H. Yang, and S. Zhang. Deep network approximation for smooth functions. *SIAM Journal on Mathematical Analysis*, 53(5):5465–5506, Jan. 2021.
[34] S. MacNamara and G. Strang. Operator splitting. In *Splitting methods in communication, imaging, science, and engineering*, pages 95–114. Springer, 2017.
[35] E. Merkurjev, T. Kostic, and A. L. Bertozzi. An mbo scheme on graphs for classification and image processing. *SIAM Journal on Imaging Sciences*, 6(4):1903–1930, 2013.
[36] V. Monga, Y. Li, and Y. C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.
[37] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.
[38] F. Qian, L. Bai, L. Cui, M. Li, H. Du, Y. Wang, and E. Hancock. Exploring the over-smoothing problem of graph neural networks for graph classification: An entropy-based viewpoint. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 2025.
[39] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.
[40] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.
[41] L. Ruiz, L. Chamon, and A. Ribeiro. Graphon neural networks and the transferability of graph neural networks. *Advances in Neural Information Processing Systems*, 33:1702–1712, 2020.
[42] L. Ruiz, Z. Wang, and A. Ribeiro. Graphon and graph neural network stability. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5255–5259. IEEE, 2021.
[43] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.
[44] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, Sept. 2008.
[45] X. Shen, H. Sun, and X. Tai. Preconditioned algorithm for difference of convex functions with applications to graph ginzburg–landau model. *Multiscale Modeling and Simulation*, 21(4):1667–1689, 2023.
[46] J. Sun, H. Li, Z. Xu, et al. Deep admm-net for compressive sensing mri. *Advances in Neural Information Processing Systems*, 29, 2016.
[47] Y. Sun, S. Wang, X. Tang, T.-Y. Hsieh, and V. Honavar. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In *Proceedings of the Web Conference 2020*, pages 673–683, 2020.
[48] X.-C. Tai, H. Liu, and R. Chan. Pottsmgnet: A mathematical explanation of encoder-decoder based neural networks. *SIAM Journal on Imaging Sciences*, 17(1):540–594, 2024.
[49] X.-C. Tai, H. Liu, R. H. Chan, and L. Li. A mathematical explanation of unet. *Mathematical Foundations of Computing*, pages 0–0, 2024.
[50] X.-C. Tai, H. Liu, L. Li, and R. H. Chan. A mathematical explanation of transformers for large language models and gpts. *arXiv preprint arXiv:2510.03989*, 2025.
[51] Y. Van Gennip, A. L. Bertozzi, et al. γ-convergence of graph ginzburg-landau functionals. *Advances in Differential Equations*, 17(11-12):1115–1180, 2012.
[52] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.
[53] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.
[54] E. Weinan, C. Ma, and L. Wu. Machine learning from a continuous viewpoint, i. *Science China Mathematics*, 63(11):2233–2266, 2020.
[55] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International*

Conference on Learning Representations, 2019.
[56] Y. Yang, J. Sun, H. Li, and Z. Xu. Deep ADMM-Net for compressive sensing mri. *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 2016.
[57] K. Yin and X.-C. Tai. An effective region force for some variational models for learning and clustering. *Journal of Scientific Computing*, 74:175–196, 2018.
[58] E. Zhang, J. Scott, Q. Du, and M. A. Porter. Ginzburg–landau functionals in the large-graph limit. *arXiv preprint arXiv:2408.00422*, 2024.
[59] K. Zhao, Q. Kang, Y. Song, R. She, S. Wang, and W. P. Tay. Adversarial robustness in graph neural networks: A hamiltonian approach. *Advances in Neural Information Processing Systems*, 36, 2024.
[60] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.

Appendix.

Appendix A. Graph and graphon operations. Let W_N be a finite graph with N nodes. We define some operations of graph functions for some $q \in [1/2, 1]$:

1. Graph gradient $\nabla_{W_N} : \mathbb{R}^N \to \mathbb{R}^{N\times N}$
$$(\nabla_{W_N}(\mathbf{u}))_{ij} := \omega_{ij}^{1-q}((\mathbf{u})_j - (\mathbf{u})_i), \quad \mathbf{u} \in \mathbb{R}^N,\ i, j = 1, \dots, N.$$
2. Graph divergence $\mathrm{div}_{W_N} : \mathbb{R}^{N\times N} \to \mathbb{R}^N$
$$(\mathrm{div}_{W_N}(U))_i := \frac{1}{2d_i}\sum_{j=1}^{N} \omega_{ij}^q (U_{ji} - U_{ij}), \quad U \in \mathbb{R}^{N\times N},\ i = 1, \dots, N.$$
3. Graph Laplacian (asymmetric) $\Delta_{W_N} : \mathbb{R}^N \to \mathbb{R}^N$
$$(\Delta_{W_N}(\mathbf{u}))_i := (\mathrm{div}_{W_N}(\nabla_{W_N}(\mathbf{u})))_i = \frac{1}{d_i}\sum_{j=1}^{N} \omega_{ij}(\mathbf{u}_i - \mathbf{u}_j). \tag{A.1}$$
4. Graph Convolution $\mathbf{h} *_W : \mathbb{R}^N \to \mathbb{R}^N$
The graph convolution characterized by a transition matrix $W \in \mathbb{R}^{N\times N}$ can be parametrized as a polynomial:
$$\mathbf{h} *_W \mathbf{u} := \sum_{p=0}^{P} h_p (W)^p \mathbf{u}$$
where $\mathbf{h} = (h_0, \dots, h_P)$ is the convolution kernel, $P > 0$ is a given integer, and $(W)^p$ denotes the matrix power. In classical GNN models, P is often chosen as 1, yielding a simple form of graph convolution:
$$\mathbf{h} *_W \mathbf{u} := h_0\mathbf{u} + h_1 W\mathbf{u}. \tag{A.2}$$
Here the matrix W can be chosen as the weight matrix W_N of a graph or its normalized version.
5. Graph inner product:
$$\langle \mathbf{u}, \mathbf{v}\rangle_{W_N} := \sum_{i=1}^{N} d_i \mathbf{u}_i \mathbf{v}_i, \quad \mathbf{u}, \mathbf{v} \in \mathbb{R}^N. \tag{A.3}$$
The norm induced by inner products $\langle \cdot, \cdot \rangle_{W_N}$ is denoted as $\| \cdot \|_{W_N}$.

6. Graph total variation:

$$\mathrm{TV}_{W_N}(\mathbf{u}) = \frac{1}{2}\sum_{i,j=1}^{N} \omega_{ij}^{q} |\mathbf{u}_i - \mathbf{u}_j|.$$

The aforementioned graph operations can also be extended to graphon functions u for $q \in [1/2, 1]$.

1. Graphon gradient $\nabla_{\mathcal{W}}$:

$$(\nabla_{\mathcal{W}} u)(x,y) := \mathcal{W}^{1-q}(x,y)(u(y) - u(x)),\ x, y \in [0,1),\ u \in L^2([0,1)).$$

2. Graphon divergence $\mathrm{div}_{\mathcal{W}}$:

$$\mathrm{div}_{\mathcal{W}}(a)(x) := \frac{1}{2d(x)} \int_0^1 \mathcal{W}^q(x,y)(a(y,x) - a(x,y))dy, \quad a \in L^2([0,1)^2).$$

3. Graphon Laplacian (asymmetric) $\Delta_{\mathcal{W}}$:

$$\Delta_{\mathcal{W}} u(x) := (\mathrm{div}_{\mathcal{W}}(\nabla_{\mathcal{W}} u))(x) = \frac{1}{d(x)} \int_0^1 \mathcal{W}(x,y)(u(x) - u(y))dy,\ u \in L^2([0,1)). \tag{A.4}$$

4. Graphon Convolution $\mathbf{h} *_{\mathcal{W}}$:
The graphon convolution is characterized by an integral operator $\mathcal{T}_{\mathcal{W}}$:

$$\mathbf{h} *_{\mathcal{W}} u := \sum_{p=0}^{P} h_p (\mathcal{T}_{\mathcal{W}})^p (u)$$

where $\mathbf{h} \in \mathbb{R}^{P+1}$ denotes the convolution kernel, $\mathcal{T}_{\mathcal{W}}(u)(x) := \int_0^1 \mathcal{W}(x,y)u(y)dy$, and $(\mathcal{T}_{\mathcal{W}})^p$ denotes the p-th power of $\mathcal{T}_{\mathcal{W}}$. When $P = 1$, the graphon convolution is simplified as

$$\mathbf{h} *_{\mathcal{W}} u := h_0 u + h_1 \mathcal{T}_{\mathcal{W}}(u). \tag{A.5}$$

5. Graphon inner product

$$\langle u, v \rangle_{\mathcal{W}} := \int_0^1 d(x)u(x)v(x)dx, \quad u, v \in L^2([0,1)). \tag{A.6}$$

6. Total variation $\mathrm{TV}_{\mathcal{W}} : L^2([0,1)) \to \mathbb{R}$

$$\mathrm{TV}_{\mathcal{W}}(u) := \frac{1}{2} \int_{[0,1)^2} \mathcal{W}^q(x,y)|u(y) - u(x)|dxdy. \tag{A.7}$$

In this work, we take $q = 1$ all the time.

Appendix B. Variational models on graphs and graphons. Variational models are fundamental mathematical tools for studying optimization problems on structured data. They have been extensively applied in various applications [9, 40, 2, 1]. Variational models can also be extended to unstructured graph data. These models typically involve minimizing energy functionals that consist of vertex-based terms and edge-based terms:

$$\widehat{\mathcal{E}}_{W_N}(\mathbf{u}) = \frac{\alpha}{N^2} \sum_{i,j=1}^{N} (W_N)_{ij}\, \phi(\mathbf{u}_i, \mathbf{u}_j) + \frac{1}{N} \sum_{i=1}^{N} F(\mathbf{u}_i), \tag{B.1}$$

where $\mathbf{u} \in \mathbb{R}^N$ defines a function on a graph with N nodes, $\phi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ measures pairwise interactions, and $F : \mathbb{R} \to \mathbb{R}$ is a data force term.

Model (B.1) can be extended to graphon $\mathcal{W}$:

$$\mathcal{E}_{\mathcal{W}}(u) = \alpha \int_0^1 \int_0^1 \mathcal{W}(x,y)\, \phi(u(x), u(y))\, dxdy + \int_0^1 d(x) F(u(x))\, dx, \tag{B.2}$$

where $u(x) \in L^2([0,1))$ is a graphon function. The connection between variational models on graphs (B.1) and graphons (B.2) arises naturally through the theory of graph limits. As mentioned in Section 2, any weight matrix of a finite graph W_N is associated with a piecewise constant graphon $\mathcal{W}_N$. Similarly, any graph function $\mathbf{u}$ is also associated with a function u_N in $L^2([0,1))$ by

$$u_N(x) = \mathbf{u}_i, \quad x \in \left[\frac{i-1}{N}, \frac{i}{N}\right)$$

for $i \in [N]$. We can easily verify the equivalence between the graph functional $\widehat{\mathcal{E}}_{W_N}$ and graphon function $\mathcal{E}_{\mathcal{W}_N}$ when acting on $\mathbf{u}$:

$$\widehat{\mathcal{E}}_{W_N}(\mathbf{u}) = \mathcal{E}_{\mathcal{W}_N}(u_N).$$

When a graph sequence $\{W_N\}_{N=1}^{\infty}$ converges to a graphon $\mathcal{W}$, it was proved in [51, 58] that the functional $\widehat{\mathcal{E}}_{W_N}$ Gamma converges to $\mathcal{E}_{\mathcal{W}}$ under mild assumptions.

Appendix C. Relaxation of the graphon Potts model and corresponding gradient flows.

We consider the soft threshold dynamics (STD) approach proposed in [31]. The original STD approach approximates the total variation in Euclidean space by $\langle 1-w, \mathcal{K} * w\rangle_{L^2}$, where w is a binary image, $\mathcal{K}$ is a Gaussian smoothing kernel, $*$ denotes the standard convolution, and $\langle\cdot,\cdot\rangle_{L^2}$ is the L^2 inner product. The STD method relaxes the binary constraint as $w \in [0,1]$ and introduces an extra entropy regularization $w\log(w)$ to the Potts model. However, this method cannot be trivially extended to graph models since there is no Gaussian kernel defined on graphs and graphons. In Euclidean spaces, the solution of the heat equation with a given initial value is exactly a convolution between the Gaussian kernel and the initial condition [18]. We can use the same idea on graph models and compute the smoothed version of a graphon function $u(x)$ by solving the initial value problem

$$\partial\hat{u}/\partial t = -\Delta_{\mathcal{W}}\hat{u}, \quad \hat{u}(x,0) = u(x), \quad x \in [0,1)$$

at $\hat{u}(x,\Delta t)$ for some small $\Delta t > 0$. A simple one-step explicit Euler scheme can give an approximation to $\hat{u}(x,\Delta t)$ by $(\mathcal{I} - \Delta t\Delta_{\mathcal{W}})u$. For simplicity, we choose $\Delta t = 1$, and extend the approximation $\langle 1-w, \mathcal{K}*w\rangle_{L^2}$ to graphon settings by

$$\langle 1-u, u - \Delta_{\mathcal{W}}u\rangle_{\mathcal{W}} = \langle 1-u, u\rangle_{\mathcal{W}} + \langle 1, -\Delta_{\mathcal{W}}u\rangle_{\mathcal{W}} + \langle u, \Delta_{\mathcal{W}}u\rangle_{\mathcal{W}}\,.$$

Here, the first term in the right-hand side $\langle 1-u, u\rangle_{\mathcal{W}}$ is 0 when $u \in \{0,1\}$. The second term $\langle 1, -\Delta_{\mathcal{W}}u\rangle_{\mathcal{W}}$ equals

$$\int_0^1 d(x)\frac{-1}{d(x)}\int_0^1 \mathcal{W}(x,y)(u(x)-u(y))dydx = \int_{[0,1)^2} \mathcal{W}(x,y)u(y)dxdy - \int_{[0,1)^2} \mathcal{W}(x,y)u(x)dxdy$$

which is also 0 because of the symmetry of $\mathcal{W}(x,y)$.

Following the STD approach, we further relax the constraint set Δ^K to

$$\widetilde{\Delta}^K := \left\{ (u_1(x), \dots, u_K(x)) \in [0,1]^K \middle| \sum_{k=1}^K u_k(x) = 1, \forall x \in [0,1) \right\},$$

and add an entropy term $\sum_{k=1}^K \langle u_k, \log(u_k) \rangle_{\mathcal{W}}$. Therefore, the relaxed version of (2.2) by the STD approach is

$$\min_U \frac{\alpha}{2} \sum_{k=1}^K \langle u_k, \Delta_{\mathcal{W}} u_k \rangle_{\mathcal{W}} + \beta \sum_{k=1}^K \langle u_k, \log(u_k) \rangle_{\mathcal{W}} + \sum_{k=1}^K \langle g_k, u_k \rangle_{\mathcal{W}} + I_{\widetilde{\Delta}^K}(U), \tag{C.1}$$

where $I_{\widetilde{\Delta}^K}$ is the indicator function of the set $\widetilde{\Delta}^K$, and $\beta > 0$ is the weight of the entropy term. Let $\mathcal{E}(U)$ be the objective functional in (C.1). For any perturbation $V = (v_1, \dots, v_K)$, we have

$$\begin{aligned} \frac{d}{d\varepsilon} \mathcal{E}(U + \varepsilon V)\Big|_{\varepsilon=0} = & \sum_{k=1}^K \frac{\alpha}{2} \left(\langle v_k, \Delta_{\mathcal{W}} u_k \rangle_{\mathcal{W}} + \langle u_k, \Delta_{\mathcal{W}} v_k \rangle_{\mathcal{W}} \right) + \beta \sum_{k=1}^K \langle 1 + \log(u_k), v_k \rangle_{\mathcal{W}} \\ & + \sum_{k=1}^K \langle g_k, v_k \rangle_{\mathcal{W}} + \left\langle \partial I_{\widetilde{\Delta}^K}(U), V \right\rangle_{\mathcal{W}}, \end{aligned}$$

where $\partial I_{\widetilde{\Delta}^K}$ denotes the subdifferential of $I_{\widetilde{\Delta}^K}$. Using the symmetry of $\Delta_{\mathcal{W}}$ with respect to $\langle \cdot, \cdot \rangle_{\mathcal{W}}$, namely

$$\langle u_k, \Delta_{\mathcal{W}} v_k \rangle_{\mathcal{W}} = \langle v_k, \Delta_{\mathcal{W}} u_k \rangle_{\mathcal{W}},$$

the diffusion contribution becomes

$$\frac{\alpha}{2} \left(\langle v_k, \Delta_{\mathcal{W}} u_k \rangle_{\mathcal{W}} + \langle u_k, \Delta_{\mathcal{W}} v_k \rangle_{\mathcal{W}} \right) = \alpha \langle v_k, \Delta_{\mathcal{W}} u_k \rangle_{\mathcal{W}}.$$

Hence

$$\frac{d}{d\varepsilon} \mathcal{E}(U + \varepsilon V)\Big|_{\varepsilon=0} = \left\langle G + \alpha \Delta_{\mathcal{W}} U + \beta + \beta \log(U) + \partial I_{\widetilde{\Delta}^K}(U), V \right\rangle_{\mathcal{W}},$$

so the variational derivative of $\mathcal{E}$ is $G + \alpha \Delta_{\mathcal{W}} U + \beta + \beta \log(U) + \partial I_{\widetilde{\Delta}^K}(U)$. We can then derive the gradient flow of (C.1) with respect to the graphon inner product as

$$\frac{\partial U}{\partial t} = - \underbrace{G}_{\text{Region force}} - \underbrace{\alpha \Delta_{\mathcal{W}} U}_{\text{Diffusion term}} - \underbrace{\left(\beta + \beta \log(U) + \partial I_{\widetilde{\Delta}^K}(U) \right)}_{\text{Nonlinear term}}. \tag{C.2}$$

Appendix D. Choices of $\mathcal{H}$ and H. For different tasks, we need to choose different $\mathcal{H}(U_T)$ for (3.1), leading to different discretizations $H(\mathbf{U}^M)$. Recall that $U_T(x) \in \mathbb{R}^K$ for any $x \in [0,1)$ and $\mathbf{U}^M \in \mathbb{R}^{N \times K}$. We use Z to denote the learnable transformation matrix in $\mathcal{H}$ and H. We give some examples of defining $\mathcal{H}$ for different tasks in Table 11. For the graph classification and node classification tasks, we use c to denote the total number of classes.

Appendix E. Sequential splitting for graphon control problems. Operator-splitting is a powerful numerical method for solving PDEs involving multiple operators, and has been used in various applications [34]. Conventional operator-splitting schemes for control problems defined

Tasks	$\mathcal{H}(U_T)$	$H(\mathbf{U}^M)$	Shape of Z
Graph classification	$Z\left(\int_{[0,1)} U_T(x)dx\right)$	$\frac{1}{N}\sum_{i=1}^N Z(\mathbf{U}^M)_i^\top$	$\mathbb{R}^{c\times K}$
Graph regression	$Z\left(\int_{[0,1)} U_T(x)dx\right)$	$\frac{1}{N}\sum_{i=1}^N Z(\mathbf{U}^M)_i^\top$	$\mathbb{R}^{1\times K}$
Node classification	$ZU_T(x)$	$\mathbf{U}^M Z^\top$	$\mathbb{R}^{c\times K}$
Node regression	$ZU_T(x)$	$\mathbf{U}^M Z^\top$	$\mathbb{R}^{1\times K}$

Table 11: Examples of $\mathcal{H}$ and H for different graph problems.

on Euclidean spaces can be directly extended to control problems on graphons. In this section, we show how to do a simple sequential splitting for the following graphon control problem:

$$\text{(E.1)} \qquad \begin{cases} u(x,0) = u_0(x), & x \in [0,1) \\ \frac{\partial u}{\partial t} = \mathcal{L}(u,x,t) + \mathcal{S}(u,x,t), & (x,t) \in [0,1)\times[0,T] \end{cases}$$

where $\mathcal{L}$ is a linear operator of u and $\mathcal{S}$ is a nonlinear operator. To solve this control system numerically, we first discretize the time domain $[0,T]$ using a mesh $\{0,h,2h,\dots,Mh\}$, where M is the total number of time steps and $h = T/M$ is the step size. We denote $t^\tau = \tau h$, and use a superscript τ to denote the variable or operator at the τ-th step. We show how to apply a sequential splitting algorithm to solve (E.1). For each $\tau = 1,\dots,M$, we proceed as follows:
Substep 1: Solve

$$\text{(E.2)} \qquad \begin{cases} \frac{\partial u}{\partial t} = \mathcal{L}(u,t) \text{ in } [0,1)\times(t^{\tau-1},t^\tau], \\ u(t^{\tau-1}) = u^{\tau-1}, \end{cases}$$

and set $u^{\tau-1/2} = u(t^\tau)$.
Substep 2: Solve

$$\text{(E.3)} \qquad \begin{cases} \frac{\partial u}{\partial t} = \mathcal{S}(u,t) \text{ in } [0,1)\times(t^{\tau-1},t^\tau], \\ u(t^{\tau-1}) = u^{\tau-1/2}, \end{cases}$$

and set $u^\tau = u(t^\tau)$.

In (E.2)-(E.3), we solve the first subproblem using a forward Euler scheme:

$$u^{\tau-1/2} = u^{\tau-1} + h\mathcal{L}^{\tau-1}(u^{\tau-1})$$

and the second subproblem by a backward Euler scheme:

$$\text{(E.4)} \qquad u^\tau = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}u^{\tau-1/2} = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}(u^{\tau-1} + h\mathcal{L}^{\tau-1}(u^{\tau-1})),$$

where $(\mathcal{I} - h\mathcal{S}^\tau)^{-1}$ denotes the solution operator of the second subproblem. The complete splitting algorithm is given in Algorithm E.1. Such a splitting algorithm has been applied to solve graph node classification problems [4]. Algorithm E.1 is first-order accurate when $\mathcal{L}$ and $\mathcal{S}$ satisfy certain conditions.

THEOREM E.1. *Let (E.1) be a control problem defined on a graphon $\mathcal{W} \in L^\infty([0,1)^2)$. Suppose $\mathcal{L}$ and $\mathcal{S}$ are third-order differentiable. Then, Algorithm E.1 is first-order accurate.*

Proof. We first derive the local truncation error for Algorithm E.1. Denote $u^{\tau-1} = u(t_{\tau-1})$, where $t_\tau = \tau h$ for $\tau = 0, \ldots, M$, and define $P_h^\tau(v) = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}(v + h\mathcal{L}^{\tau-1}(v))$ for any $v \in L^\infty([0,1))$. We get

$$P_h^\tau(u(t_{\tau-1})) = (\mathcal{I} + h\mathcal{S}^\tau + (h\mathcal{S}^\tau)^2)(u(t_{\tau-1}) + h\mathcal{L}^{\tau-1}(u(t_{\tau-1}))) + O(h^3).$$

Since $\mathcal{S}^\tau(u(t_{\tau-1}) + h\mathcal{L}^{\tau-1}(u(t_{\tau-1})))$ can be expanded as

$$\left(\mathcal{I} + h\mathcal{L}^{\tau-1}(u(t_{\tau-1}))\frac{\partial}{\partial u} + h\frac{\partial}{\partial t}\right)\mathcal{S}^{\tau-1}(u(t_{\tau-1})) + O(h^2),$$

we see that

$$\begin{aligned} P_h^\tau(u(t_{\tau-1})) &= u(t_{\tau-1}) + h\mathcal{L}^{\tau-1}(u(t_{\tau-1})) + h\mathcal{S}^{\tau-1}(u(t_{\tau-1})) \\ &\quad + h^2(\mathcal{I} + \mathcal{L}^{\tau-1}(u(t_{\tau-1}))\frac{\partial}{\partial u} + \frac{\partial}{\partial t})\mathcal{S}^{\tau-1}(u(t_{\tau-1})) + O(h^3). \end{aligned} \tag{E.5}$$

We also expand $u(t_\tau)$ at $t = t_{\tau-1}$ directly as

$$u(t_\tau) = \left(\mathcal{I} + h\frac{\partial}{\partial t} + \frac{h^2}{2}\frac{\partial^2}{\partial t^2}\right)u(t_{\tau-1}) + O(h^3). \tag{E.6}$$

By subtracting (E.5) from (E.6), we get

$$u(t_\tau) - P_h^\tau(u(t_{\tau-1})) = O(h^2).$$

Since $\mathcal{W} \in L^\infty([0,1)^2)$, we also get

$$\|u(t_\tau) - P_h^\tau(u(t_{\tau-1}))\|_{\mathcal{W}}^2 = O(h^2).$$

Suppose $\{u^\tau\}_{\tau=1}^M$ is the sequence of numerical solutions generated by Algorithm E.1 such that $u^\tau = P_h^\tau(u^{\tau-1})$. Then, we have

$$\begin{aligned} &\|u(t_\tau) - u^\tau\|_{\mathcal{W}}^2 \le \|P_h^\tau(u(t_{\tau-1})) - P_h^\tau(u^{\tau-1})\|_{\mathcal{W}}^2 + O(h^2) \\ &= \|u(t_{\tau-1}) + h\mathcal{L}^{\tau-1}(u(t_{\tau-1})) + h\mathcal{S}^{\tau-1}(u(t_{\tau-1})) - u^{\tau-1} - h\mathcal{L}^{\tau-1}(u^{\tau-1}) - h\mathcal{S}^{\tau-1}(u^{\tau-1})\|_{\mathcal{W}}^2 + O(h^2). \end{aligned}$$

Because of the Lipschitz continuity of $\mathcal{S}$ and $\mathcal{L}$, there exists a constant $c_L > 0$ such that

$$\|\mathcal{L}^{\tau-1}(u(t_{\tau-1})) + \mathcal{S}^{\tau-1}(u(t_{\tau-1})) - \mathcal{L}^{\tau-1}(u^{\tau-1}) - \mathcal{S}^{\tau-1}(u^{\tau-1})\|_{\mathcal{W}}^2 \le c_L\|u(t_{\tau-1}) - u^{\tau-1}\|_{\mathcal{W}}^2.$$

Consequently, we derive that

$$\begin{aligned} \|u(t_M) - u^M\|_{\mathcal{W}}^2 &\le (1 + c_L h)\|u(t_{M-1}) - u^{M-1}\|_{\mathcal{W}}^2 + O(h^2) \\ &\le (1 + c_L h)^M\|u(t_0) - u^0\|_{\mathcal{W}}^2 + O(Mh^2) \\ &= O(h). \end{aligned}$$

∎

For control problems involving multiple operators, we may choose a more complicated splitting algorithm, such as the hybrid splitting algorithm introduced in [48], to solve them. If we consider the nonlinear operator $(\mathcal{I} - h\mathcal{S}^\tau)^{-1}$ as an activation function and $\mathcal{L}^\tau$ as a learnable linear operator,

Algorithm E.1 A sequential splitting method to solve (E.1)

Data: The initial condition u_0.
Result: The computed solution $u(x,T)$.
Set $u^0 = u_0$.
for $\tau = 1, \cdots, M$ **do**
Compute

$$u^\tau = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}(u^{\tau-1} + h\mathcal{L}^{\tau-1}(u^{\tau-1})).$$

end for

equation (E.4) is then a continuous counterpart of a plain graph convolutional layer [25]. Such connections have been extensively explored for various neural networks defined on structured grids [10, 27, 48, 49].

Appendix F. Two hybrid splitting algorithms. In this section, we consider the following control problem defined on a graphon $\mathcal{W}$:

$$\text{(F.1)} \qquad \begin{cases} u(x,0) = u_0(x), & x \in [0,1) \\ \frac{\partial u}{\partial t} = \mathcal{L}_1(u,x,t) + \mathcal{L}_2(u,x,t) + \hat{\mathcal{S}}(u,x,t) + \mathcal{S}(u,x,t), & (x,t) \in [0,1) \times [0,T] \end{cases}$$

We consider the following two splitting algorithms to solve (F.1).

Algorithm F.1 The first hybrid splitting algorithm to solve (F.1)

Input: The initial condition $u(x,0)$.
Output: The computed solution $u(x,T)$.
Set $u^0 = u(x,0)$.
for $\tau = 1, \cdots, M$ **do**
Solve $u^{\tau-\frac{1}{2},1} = (\mathcal{I} - 2h\hat{\mathcal{S}}^\tau)^{-1}(u^{\tau-1} + 2h\mathcal{L}_1^{\tau-1}(u^{\tau-1}))$
Solve $u^{\tau-\frac{1}{2},2} = u^{\tau-1} + 2h\mathcal{L}_2^{\tau-1}(u^{\tau-1})$
Solve $u^\tau = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}(\frac{1}{2}(u^{\tau-\frac{1}{2},1} + u^{\tau-\frac{1}{2},2}))$
end for

Algorithm F.2 The second hybrid splitting algorithm to solve (F.1)

Data: The initial condition $u(x,0)$.
Result: The computed solution $u(x,T)$.
Set $u^0 = u(x,0)$.
for $\tau = 1, \cdots, M$ **do**
Solve $u^{\tau-\frac{1}{2},1} = (\mathcal{I} - h\hat{\mathcal{S}}^\tau)^{-1}(u^{\tau-1} + h\mathcal{L}_1^{\tau-1}(u^{\tau-1}))$
Solve $u^{\tau-\frac{1}{2},2} = u^{\tau-\frac{1}{2},1} + h\mathcal{L}_2^{\tau-1}(u^{\tau-\frac{1}{2},1})$
Solve $u^\tau = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}(u^{\tau-\frac{1}{2},2})$
end for

THEOREM F.1. *Let (F.1) be a control problem defined on a graphon $\mathcal{W} \in L^\infty([0,1)^2)$. Suppose $\mathcal{L}_1$, $\mathcal{L}_2$, $\hat{\mathcal{S}}$, and $\mathcal{S}$ are third-order differentiable. Then, Algorithm F.1 is first-order accurate.*

Proof. We first derive the local truncation error for Algorithm F.1. Denote $u^{\tau-1} = u(t_{\tau-1})$, where $t_\tau = \tau h$ for $\tau = 0, \dots, M$. For Algorithm F.1, the numerical solution operator is

$$P^\tau_{h,1}(v) = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}\left(\frac{1}{2}(\mathcal{I} - 2h\hat{\mathcal{S}}^\tau)^{-1}(v + 2h\mathcal{L}_1^{\tau-1}(v)) + \frac{1}{2}(v + 2h\mathcal{L}_2^{\tau-1}(v))\right).$$

Using Taylor expansion as in Theorem E.1, we have

$$\begin{aligned}(\mathcal{I} - 2h\hat{\mathcal{S}}^\tau)^{-1}(v + 2h\mathcal{L}_1^{\tau-1}(v)) &= (\mathcal{I} + 2h\hat{\mathcal{S}}^\tau)(v + 2h\mathcal{L}_1^{\tau-1}(v)) + O(h^2)\\ &= v + 2h\mathcal{L}_1^{\tau-1}(v) + 2h\hat{\mathcal{S}}^\tau(v) + O(h^2).\end{aligned}$$

Therefore, the term inside the outer inverse is

$$\begin{aligned}&\frac{1}{2}\Big(v + 2h\mathcal{L}_1^{\tau-1}(v) + 2h\hat{\mathcal{S}}^\tau(v)\Big) + \frac{1}{2}\Big(v + 2h\mathcal{L}_2^{\tau-1}(v)\Big) + O(h^2)\\ &= v + h\mathcal{L}_1^{\tau-1}(v) + h\hat{\mathcal{S}}^\tau(v) + h\mathcal{L}_2^{\tau-1}(v) + O(h^2).\end{aligned}$$

Applying the outer operator $(\mathcal{I} - h\mathcal{S}^\tau)^{-1} = \mathcal{I} + h\mathcal{S}^\tau + O(h^2)$, we get

$$P^\tau_{h,1}(u(t_{\tau-1})) = u(t_{\tau-1}) + h(\mathcal{L}_1^{\tau-1} + \mathcal{L}_2^{\tau-1} + \hat{\mathcal{S}}^\tau + \mathcal{S}^\tau)(u(t_{\tau-1})) + O(h^2).$$

Expanding $u(t_\tau)$ at $t = t_{\tau-1}$ using the exact PDE (F.1) gives

$$\begin{aligned}u(t_\tau) &= u(t_{\tau-1}) + h\frac{\partial u}{\partial t}\Big|_{t=t_{\tau-1}} + O(h^2)\\ &= u(t_{\tau-1}) + h(\mathcal{L}_1^{\tau-1} + \mathcal{L}_2^{\tau-1} + \hat{\mathcal{S}}^{\tau-1} + \mathcal{S}^{\tau-1})(u(t_{\tau-1})) + O(h^2).\end{aligned}$$

Subtracting the two and using the smoothness of the operators with respect to t, we obtain

$$\|u(t_\tau) - P^\tau_{h,1}(u(t_{\tau-1}))\|^2_{\mathcal{W}} = O(h^2).$$

By the Lipschitz continuity of the operators, following the same induction steps as in Theorem E.1, we conclude that $\|u(t_M) - u^M\|^2_{\mathcal{W}} = O(h)$. Thus Algorithm F.1 is first-order accurate. □

THEOREM F.2. *Let (F.1) be a control problem defined on a graphon $\mathcal{W} \in L^\infty([0,1)^2)$. Suppose $\mathcal{L}_1$, $\mathcal{L}_2$, $\hat{\mathcal{S}}$, and $\mathcal{S}$ are third-order differentiable. Then, Algorithm F.2 is first-order accurate.*

Proof. We derive the local truncation error for Algorithm F.2. Denote $u^{\tau-1} = u(t_{\tau-1})$, where $t_\tau = \tau h$ for $\tau = 0, \dots, M$. For Algorithm F.2, the numerical solution operator is

$$P^\tau_{h,2}(v) = (\mathcal{I} - h\mathcal{S}^\tau)^{-1}\Big(\tilde{v} + h\mathcal{L}_2^{\tau-1}(\tilde{v})\Big),$$

where $\tilde{v} = (\mathcal{I} - h\hat{\mathcal{S}}^\tau)^{-1}(v + h\mathcal{L}_1^{\tau-1}(v))$. Standard Taylor expansion gives

$$\tilde{v} = v + h\mathcal{L}_1^{\tau-1}(v) + h\hat{\mathcal{S}}^\tau(v) + O(h^2).$$

Substituting $\tilde{v}$ back yields

$$\begin{aligned}P^\tau_{h,2}(v) &= (\mathcal{I} + h\mathcal{S}^\tau)\Big(v + h\mathcal{L}_1^{\tau-1}(v) + h\hat{\mathcal{S}}^\tau(v) + h\mathcal{L}_2^{\tau-1}(v)\Big) + O(h^2)\\ &= v + h(\mathcal{L}_1^{\tau-1} + \hat{\mathcal{S}}^\tau + \mathcal{L}_2^{\tau-1} + \mathcal{S}^\tau)(v) + O(h^2).\end{aligned}$$

This matches the expansion for Algorithm F.1 up to $O(h^2)$. Thus, the local truncation error is again $O(h^2)$, and following the exact same steps, the global error is bounded by $O(h)$. Hence, Algorithm F.2 is also first-order accurate. □

Appendix G. Model implementation details and computational cost.

This section summarizes the implementation details of all models used in numerical experiments in Section 6. For all experiments, the compared message-passing models are GCN, GraphSAGE, Graph Transformer (GT), and GIN, implemented using their standard DGL implementations [53]. The hyperparameters for these models and corresponding computational time are summarized in Table 12. The total parameter counts listed here do not include the task-dependent head H. The structure of the head module is identical for all models when solving the same problem.

For the two proposed GraphPottsNet models, GPN(par) and GPN(seq), we implement them as described in Section 5. We use the finite-graph discretizations of the two hybrid splitting schemes. Unless otherwise specified, we set the number of unrolled steps to $M = 4$, terminal time to $T = 1$, and regularization parameter to $\alpha = 1$. The first layer is a standard graph convolution layer for lifting the input feature dimension to the specified hidden dimension (also called the width), and the last layer is the task-dependent readout/transformation layer described in Table 11. For the node classification task, the region force network is implemented as a GraphSAGE network, $\hat{\sigma}_1$ and $\hat{\sigma}_2$ are directly implemented as a tanh activation, and the convolution term $\mathcal{A}^{\tau-1}$ is implemented as a SAGEConv (4.15). For the other tasks, the region force network is implemented as a GIN network. $\hat{\sigma}_1$ is implemented as a tanh activation, $\hat{\sigma}_2$ is parameterized by a learnable two-layer MLP, and we use the GIN-type graph convolution in (4.17) for $A^{\tau-1}$. The hyperparameters for the proposed GraphPottsNet models and corresponding computational time are summarized in Table 13. Here, we also list in Table 14 the number of parameters for each trainable module in our proposed GPN, using the MNIST superpixel graph classification as an example. Here, in each layer, we use (4.17) to parameterize $\hat{\sigma}_2^\tau(\mathbf{U}^{\tau-1} + A^{\tau-1}(\mathbf{U}^{\tau-1}))$, so all learnable parameters are in $\hat{\sigma}_2^\tau$, and parameters in $A^{\tau-1}$ are fixed.

In all cases, we use an Adam optimizer with a learning rate of 0.01 to train all models for 200 epochs, and use the validation set accuracy to select the model with the best validation performance, which is then used for test evaluation. Typically, the training can be stopped when the model performance on the validation set stops improving. In practice, we found that 200 epochs is sufficient for all models to reach the early stopping point. Even if we continue for more training epochs, the models will just overfit the training data. The training and testing time are measured on a single NVIDIA V100-32GB GPU, and the memory used is the maximum GPU memory used during training.

In our proposed GPN models, we have several hyperparameters to choose, including the depth M, the terminal time T, hidden dimension c_h, and the regularization parameter α. In all our experiments, we fixed $M = 2$ and $T = 1$ on the citation network dataset and $M = 4$ and $T = 1$ for all other datasets. Typically, increasing M would lead to a deeper GNN. However, deep GNN models may suffer from over-smoothing issues [38]. Therefore, we suggest $M = 2$ for node classification tasks and $M = 4$ for other tasks as the default choice. For the hidden dimension c_h, a larger c_h can potentially improve the model's capacity, but it also increases the computational cost, so it can be chosen depending on the computational budget. For the regularization parameter α, it depends on the noise level of the data. A larger α can lead to a stronger regularization effect, which can be beneficial when the noise level is high. Therefore, we suggest using $\alpha = 1$ as a default choice for α, and adaptively increasing α when the noise level of the data is high. One can also choose the suitable α by cross-validation.

Task	Dataset	Model	Depth	Width	Total parameters	Training time (per epoch)/ testing time	GPU memory usage
Graph regression	QM9	GCN	4	120	44880	177.8s/45.3s	38.1Mb
		GraphSAGE	4	90	50220	180.1s/45.8s	35.6Mb
		GT	4	48	51600	178.3s/44.7s	44.9Mb
		GIN	4	100	53520	269.4s/66.9s	79.4Mb
Node classification	CORA	GCN	1	32	45888	8.1×10^{-3}s/3.4×10^{-3}s	93.9Mb
		GraphSAGE	1	16	45872	6.6×10^{-3}s/2.6×10^{-3}s	76.0Mb
		GT	1	32	54720	2.8×10^{-1}s/6.5×10^{-2}s	101.5Mb
		GIN	1	32	47008	7.5×10^{-3}s/3.0×10^{-3}s	122.6Mb
Node classification	CITESEER	GCN	1	32	118528	7.7×10^{-3}s/3.3×10^{-3}s	255.4Mb
		GraphSAGE	1	16	118512	8.4×10^{-3}s/3.5×10^{-3}s	244.7Mb
		GT	1	32	12736	4.2×10^{-1}s/1.3×10^{-1}s	264.5Mb
		GIN	1	32	119648	7.5×10^{-3}s/3.1×10^{-3}s	348.1Mb
Node classification	PUBMED	GCN	1	32	16032	8.0×10^{-3}s/3.4×10^{-3}s	228.8Mb
		GraphSAGE	1	16	16016	6.7×10^{-3}s/2.6×10^{-3}s	172.8Mb
		GT	1	32	24864	5.6s/2.5s	291.0Mb
		GIN	1	32	17152	7.8×10^{-3}s/3.1×10^{-3}s	302.1Mb
Graph classification	MUTAG	GCN	4	120	45240	2.1×10^{-1}s/5.2×10^{-2}s	25.3Mb
		GraphSAGE	4	90	50760	2.1×10^{-1}s/5.1×10^{-2}s	24.1Mb
		GT	4	48	53640	5.4×10^{-1}s/2.0×10^{-2}s	30.3Mb
		GIN	4	100	51900	2.0×10^{-1}s/5.0×10^{-2}s	27.93Mb
Graph classification	NCI1	GCN	4	120	48840	4.5s/1.1s	34.1Mb
		GraphSAGE	4	90	56160	4.4s/1.1s	31.0Mb
		GT	4	48	54840	12.2s/2.9s	43.0Mb
		GIN	4	100	54900	4.5s/1.1s	41.0Mb
Graph classification	IMDBBINARY	GCN	4	120	44520	1.2s/2.7×10^{-1}s	29.3Mb
		GraphSAGE	4	90	49680	1.0s/2.5×10^{-1}s	28.2Mb
		GT	4	48	53400	2.3s/5.8×10^{-1}s	45.8Mb
		GIN	4	100	51300	1.1s/2.6×10^{-1}s	31.6Mb
Graph classification	ENZYMES	GCN	4	120	46560	7.0×10^{-1}s/2.0×10^{-1}s	35.2Mb
		GraphSAGE	4	90	52740	6.8×10^{-1}s/1.7×10^{-1}s	32.9Mb
		GT	4	48	54080	2.0s/5.9×10^{-1}s	49.4Mb
		GIN	4	100	53000	6.7×10^{-1}s/1.7×10^{-1}s	39.8Mb
Graph classification	MNIST	GCN	4	120	44760	96.2s/15.3s	101.1Mb
		GraphSAGE	4	90	50040	93.2s/16.1s	91.6Mb
		GT	4	40	53480	700.4s/116.0s	209.2Mb
		GIN	4	100	51500	95.0s/16.5s	118.6Mb

Table 12: Implementation details and hyperparameters for baseline message-passing models used in the numerical experiments.

Task	Dataset	Model	M/c_h	RF (depth/Width)	Total parameters	Training time (per epoch)/ testing time	GPU memory usage
Graph regression	QM9	GPN(par)	4/60	GIN(4/60)	49020	183s/45.4s	53Mb
		GPN(seq)	4/60	GIN(4/60)	49020	186s/46.3s	55Mb
Node classification	CORA	GPN(par)	2/16	GCN(1/16)	46944	1.7×10^{-2}s/6.3×10^{-3}s	125.5Mb
		GPN(seq)	2/16	GCN(1/16)	46944	1.8×10^{-2}s/6.5×10^{-3}s	126.1Mb
Node classification	CITESEER	GPN(par)	2/16	GCN(1/16)	119584	1.8×10^{-2}s/6.8×10^{-3}s	351.9Mb
		GPN(seq)	2/16	GCN(1/16)	119584	1.9×10^{-2}s/6.7×10^{-3}s	352.7Mb
Node classification	PUBMED	GPN(par)	2/16	GCN(1/16)	17088	1.6×10^{-2}s/6.1×10^{-3}s	324.4Mb
		GPN(seq)	2/16	GCN(1/16)	17088	1.7×10^{-2}s/6.4×10^{-3}s	328.7Mb
Graph classification	MUTAG	GPN(par)	4/60	GIN(4/60)	49380	2.3×10^{-1}s/5.4×10^{-2}s	32.5Mb
		GPN(seq)	4/60	GIN(4/60)	49380	2.2×10^{-1}s/5.5×10^{-2}s	34.2Mb
Graph classification	NCI1	GPN(par)	4/60	GIN(4/60)	52980	4.7s/1.1s	47.7Mb
		GPN(seq)	4/60	GIN(4/60)	52980	4.8s/1.1s	50.8Mb
Graph classification	IMDBBINARY	GPN(par)	4/60	GIN(4/60)	48660	1.1s/2.7×10^{-1}s	37.4Mb
		GPN(seq)	4/60	GIN(4/60)	48660	1.1s/2.7×10^{-1}s	38.9Mb
Graph classification	ENZYMES	GPN(par)	4/60	GIN(4/60)	50700	7.2×10^{-1}s/1.7×10^{-1}s	48.4Mb
		GPN(seq)	4/60	GIN(4/60)	50700	7.3×10^{-1}s/1.7×10^{-1}s	51.8Mb
Graph classification	MNIST	GPN(par)	4/60	GIN(4/60)	48420	96.7s/16.4s	157.5Mb
		GPN(seq)	4/60	GIN(4/60)	48420	98.3s/17.0s	170.9Mb

Table 13: Implementation details and hyperparameters for the proposed GraphPottsNet variants. "RF" denotes the parameterization of G_θ used in each setting.

Trainable module	Description	Number of parameters
U^0	Initialization operator	240
G_θ	The region force network	18900
$\hat{\sigma}_2^\tau$	The MLP at each layer, $\tau = 1, 2, 3, 4$	7320
Total	All trainable modules listed above	48420

Table 14: Parameter decomposition of the proposed GPN used for the MNIST superpixel graph classification experiment.