TIDE: Time Derivative Diffusion for Deep Learning on Graphs

Maysam Behmanesh [*1] **Maximilian Krahn** [*1 2] **Maks Ovsjanikov** [1]

Abstract

A prominent paradigm for graph neural networks is based on the message-passing framework. In this framework, information communication is realized only between neighboring nodes. The challenge of approaches that use this paradigm is to ensure efficient and accurate *long-distance communication* between nodes, as deep convolutional networks are prone to oversmoothing. In this paper, we present a novel method based on time derivative graph diffusion (TIDE) to overcome these structural limitations of the message-passing framework. Our approach allows for optimizing the spatial extent of diffusion across various tasks and network channels, thus enabling medium and long-distance communication efficiently. Furthermore, we show that our architecture design also enables local message-passing and thus inherits from the capabilities of local message-passing approaches. We show that on both widely used graph benchmarks and synthetic mesh and graph datasets, the proposed framework outperforms state-of-the-art methods by a significant margin.[+]

1. Introduction

Designing efficient and scalable architectures for learning on graphs is a central problem in machine learning with applications in a broad range of disciplines, including data mining (Li et al., 2019b; Zhang et al., 2019), recommendation systems (Zhang et al., 2019), text classification (Yao et al., 2019), image analysis and matching (Sarlin et al., 2020) and even molecular property prediction (Wieder et al., 2020) among myriad others.

A very wide variety of graph neural network (GNN) approaches have been proposed over the past several years (see, e.g., (Zhou et al., 2020; Wu et al., 2020) for recent surveys), ranging from spectral methods, spatial or convolutional designs, recurrent graph neural networks, or graph auto-encoders as well as many other hybrid techniques. A particularly prominent and widely-used category of approaches is given by the convolutional graph neural networks, and especially those based on message-passing, following the design introduced in (Kipf & Welling, 2017) and extended significantly in many follow-up works, e.g., (Li et al., 2018b; Zhuang & Ma, 2018; Chamberlain et al., 2021b; Thorpe et al., 2021).

The key strengths of convolutional graph neural networks, as introduced in (Kipf & Welling, 2017), include their simplicity and computational efficiency, their ability to be composed with other neural networks as well as their ability to generalize across different graphs (i.e., learning weights that could be applied on unseen graphs). As a result, the original GCN approach (Kipf & Welling, 2017) is still highly effective and is widely used in many applications.

Nevertheless, a prominent limitation of message-passing approaches, such as GCN and related methods is *oversmoothing*, which implies that such networks tend to be difficult to train beyond a small number of layers (Oono & Suzuki, 2019). Furthermore, since typical message-passing operators only ensure communication between nodes within a 1-hop neighborhood, this means that message-passing approaches can hinder *long-distance information propagation*, which can limit their utility in scenarios, where such long-range communication is important.

In this work, we demonstrate that a simple modification to the standard GCN design can be used to enable information propagation across possibly distant nodes within a shallow, one layer graph neural network, which does not have the oversmoothing issues of traditional deep GNNs. Our design inherits most of the advantages of standard message-passing methods, including computational efficiency, their domain independence, and their ability to generalize across different graphs.

Key to our approach is our use of *learnable time diffusion* which allows information propagation on the graph

[*]Equal contribution [1]LIX, École polytechnique, IP Paris, France [2]Aalto University, Finland. Correspondence to: Maysam Behmanesh <maysam.behmanesh@lix.polytechnique.fr>, Maximilian Krahn <maximilian.krahn@icloud.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[+]Our implementation is available at https://github.com/maysambehmanesh/TIDE

while being able to optimize the communication extent in a task-dependent manner. Our method is inspired by recent approaches in surface learning (Sharp et al., 2022) that have introduced the notion of *learnable time diffusion* as a way to replace convolution for information sharing when learning on surfaces. However, unlike the method presented in (Sharp et al., 2022), which explicitly aimed at robustness to changes in connectivity and used standard heat diffusion, we base our approach on learnable *time-derivative* diffusion. As we demonstrate in this work, this allows us to retain the advantages of the message-passing framework, and ensure both local ($1-2$ hop neighborhoods) and possibly global (n-hop neighborhoods, up to the whole graph) information communication in a single an efficiently learnable architecture. To summarize, our key contributions include:

1. We introduce *time-derivative diffusion* as an effective mechanism for information propagation within graph neural networks.
2. We propose an architecture based on time-derivative diffusion, which enables both local and global information propagation, in a differentiable manner, while generalizing the standard message-passing framework.
3. With this mechanism at hand, we develop a simple and scalable architecture that outperforms strong baselines on several benchmarks.

Our method is particularly useful on either sparsely labeled graphs or in scenarios where longer dependencies are important to capture global structures on graphs. Our method is easy to train and does not require a significant computational overhead in either learning time or memory footprint. Finally, while we focus on the GCN and derived architectures, we believe that the use of learnable time-derivative diffusion can be a broadly useful tool in graph neural networks and can, in the future, be combined with other architecture designs.

2. Related Work

Graph neural networks The key goal of Graph Neural Networks (GNNs) is to compute the representation of all unlabeled nodes or edges. To achieve this, many network designs employ a message-passing approach to exchange information of each node or edge with its neighbors until reaching the equilibrium state (Scarselli et al., 2009). Based on the approach, GNNs can perform convolution using two types of models: spatial or spectral. Spatial GNNs perform message passing directly by considering neighborhood structure in the graphs (Wu et al., 2020). Below we list several well-known spatial GNNs: GraphSage uses an aggregation function to represent each node by aggregate results of its neighborhood (Hamilton et al., 2017), Message Passing neural network (MPNN) runs K-step message-passing iterations to let information propagate further (Gilmer et al., 2017), and Graph Attention Networks (GAT) use an attention mechanism to combine different contributions of neighboring nodes (Veličković et al., 2017). One of the most prominent models for message passing is the Graph Convolutional Network (GCN) (Kipf & Welling, 2017). GCN simplifies ChebNets architecture (Defferrard et al., 2016) by using filters operating on the 1-hop neighborhoods of the graph. Within GCN, the features from the neighbors get passed to the node by convolution layers.

Spectral GNNs perform convolution by transforming node representations into the spectral domain using the graph Fourier transform (Wu et al., 2020). GNN-ARMA (Bianchi et al., 2021) uses an autoregressive moving average (ARMA) filter to capture the global graph structures. To implement an efficient convolution on the graph, (Xu et al., 2019; Behmanesh et al., 2022) consider the convolution via wavelet transform instead of Fourier transform by taking the graph wavelet as a set of bases of spectral GNN.

A common problem on graph neural networks is their oversmoothing behavior, which hinders their expressive power when increasing the number of layers, (Li et al., 2018a). Multiple approaches have tried to solve this issue, e.g., by using different co-training strategies (Li et al., 2018a), changing the architecture by adding different kinds of residual connections (Li et al., 2019a), a PageRank-based propagation schema (Klicpera et al., 2018) or a separate analysis of propagation and representation of the features (Liu et al., 2020).

Learned diffusion / learned ODEs A stepping stone in deep neural networks was interpreting these as neural ordinary differential equations (ODE) (Chen et al., 2018). This idea has been further expanded in numerous works such as (Dupont et al., 2019; Finlay et al., 2020; Li et al., 2020; Liu et al., 2019). For graph neural networks, this interpretation has been applied in various works, for instance in (Avelar et al., 2019; Poli et al., 2019; Xhonneux et al., 2020). Additionally, ODEs have not only been used for graph neural networks but also for shape learning. In geometric learning, Sharp et al. introduced a diffusion on shapes with the learnable time parameter t (Sharp et al., 2022).

Graph diffusion process GRAND (Chamberlain et al., 2021b) is a prominent graph diffusion work that interprets graph convolution networks as a solution to the heat diffusion equation. This work forms the basis for models such as GRAND++ (Thorpe et al., 2021), which uses an additional source term, and BLEND (Chamberlain et al., 2021a), which uses additional Beltrami features. However, in these models, the diffusion time is treated as a fixed hyperparameter and is not learnable. This limits the flexibility and

adaptability of such models to different graph structures.

Long range dependency Commonly-used message-passing GNNs do not accommodate long-distance communication, which limits their expressivity to a small neighborhood around the node. In (Alon & Yahav, 2020) a novel explanation for training is introduced to prevent over-squashing in GNNs from long-range patterns in the data. In (Abu-El-Haija et al., 2019), these relationships are learned by repeatedly mixing feature representations of neighbors at various distances.

3. Background, Motivation & Overview

Our work builds upon the successful paradigm in graph neural networks based on message passing. Methods within this framework, pioneered by the GCN architecture (Kipf & Welling, 2017) and its variants (Bianchi et al., 2021; Chen et al., 2020) are based on two key components. First, an information message passing operator L is assembled, typically using a normalized Laplacian or adjacency matrix. Second, this operator is used, jointly with a non-linearity function σ, to construct a single layer of the graph neural network with learnable linear weights W. Such layers can then be stacked into a deep multi-layer network by iterating the action of a single layer and using separate learnable weights at each level.

Given a graph consisting of n nodes, let $U \in \mathbb{R}^{n\times m}$ be a matrix of m scalar fields $u \in \mathbb{R}^n$ representing, for example, some feature values at the nodes. The most basic variant of this approach can be summarized via the following formula:

$$\mathcal{N}_{\mathcal{W}}(U) = \mathcal{L}_k \circ \mathcal{L}_{k-1} ... \circ \mathcal{L}_1 \circ \mathcal{L}_0(U). \tag{1}$$

Here U is some set of input features, $\mathcal{L}_k$ is the kth layer of the neural network, and $\mathcal{W}$ denotes the set of all learnable weights, which are composed of weights associated with every layer. In particular, a typical layer $\mathcal{L}_k$ has the form:

$$\mathcal{L}_k(U) = \sigma\left(LUW^k\right). \tag{2}$$

where σ is some non-linearity, W^k is a matrix of learnable weights at layer k and L is a message passing operator. For example, L can be the standard graph Laplacian matrix or the normalized adjacency matrix with self-loops as used in (Kipf & Welling, 2017).

While simple and efficient, this approach has several key drawbacks. Perhaps the most prominent limitation is the well-known *oversmoothing* effect of the basic graph neural network architecture. This effect implies that networks of the type described in Eq. (1) tend to saturate very quickly, even for small to moderate k. In other words, it is difficult to build networks that are both easy to train and have a significant depth.

Since the most commonly-used message-passing operators such as the graph Laplacian or its normalized variants only enable information propagation within the 1-hop neighborhood of each node, this means that standard graph neural networks do not easily enable *long-distance information communication*, which can limit their utility in practice. Unfortunately, simple strategies such as expanding the receptive field size of the message passing operator also have limited success.

Motivation Our work aims to address the issue raised above and is inspired by recent approaches that exploit properties of the *diffusion* process to enable communication on both graphs (Chamberlain et al., 2021b; Thorpe et al., 2021) and more general domains (Sharp et al., 2022). Specifically, in (Chamberlain et al., 2021b) the authors showed that information propagation within a graph neural network can be formulated from the perspective of anisotropic diffusion, which furthermore encompasses and generalizes the standard message-passing formulation. However, in that work, the diffusion time was still used as a fixed hyperparameter (set to 1).

On the other hand, with the computer graphics community (Sharp et al., 2022) it has recently been shown that diffusion with *a learnable time* parameter can be used to enable information propagation on geometric domains. Moreover, this process can adapt the receptive field size of different channels from local to global depending on the task. In particular, rather than using message passing, the key idea in (Sharp et al., 2022) is to ensure information propagation by using the diffusion equation:

$$\frac{\partial u}{\partial t} = -\Delta u, \tag{3}$$

where Δ is a positive semi-definite Laplacian operator. The solution to the diffusion equation is given by the heat operator:

$$u_0 \to u_t, \text{ where } u_t = H_t(u_0) \tag{4}$$

Thus, the idea advocated in (Sharp et al., 2022) is to use Eq. (4) to build a layer, where each signal u is propagated for some task-dependent, *learnable time* t. Notably, the heat operator H_t has a closed form expression and is given by the operator (matrix) exponential $H_t = \exp(-t\Delta)$ and can moreover be approximated using the Laplacian spectral basis for efficient computation.

On the other hand, the architecture design of the approach in (Sharp et al., 2022) 9s focused on ensuring robustness to significant discretization changes, especially on triangle meshes. This means that the connectivity structure of the underlying graph is not explicitly relied upon. In contrast, within graph neural network applications, the connectivity structure is often crucial and is a major source of useful

information, which, in part, also explains the success of message-passing approaches.

Our main goal thus is to combine the local accuracy and sensitivity to the graph structure of message-passing approaches, with the ability to learn the receptive field size and ensure global information propagation without oversmoothing, enabled by diffusion with learnable time parameters.

Overview To achieve these goals, we propose a novel model, which combines learnable diffusion with message passing in a single principled framework while being efficient and achieving accurate results on a wide range of benchmarks.

Our method is based on the diffusion equation, Eq. (3), similar to the approach in (Sharp et al., 2022). However, rather than using diffusion itself to ensure information propagation on the graph, we propose to use *time-derivative diffusion*. That is, rather than using Eq (4) to propagate information on a graph within a graph neural network, we propose to use the following time derivative formulation instead:

$$u_0 \to -\frac{\partial u_t}{\partial t}, \text{ where } u_t = H_t(u_0). \tag{5}$$

Our key idea, therefore, is to use Eq. (5) to enable information communication in a graph within a graph neural network. Despite a relatively simple change, as we demonstrate below, this formulation has several distinguishing characteristics. First, conceptually, time-derivative diffusion is closely linked to wavelets, since, for example, it is well known that the derivative in time of the heat kernel, which is simply a Gaussian in Euclidean space, corresponds exactly to the Mexican hat wavelet (Hou & Qin, 2012; Kirgo et al., 2021). Moreover, and more importantly, as we demonstrate below, unlike standard diffusion, a time-derivative-based formulation allows us to retain the power of local message-passing approaches while, at the same time, enabling long-distance communication without oversmoothing.

4. Method

4.1. Time Derivative Diffusion

As mentioned above, in the continuous setting, the diffusion process is described as the solution of the heat equation, Eq. (3): $\frac{d}{dt}u_t = Lu_t$, In this equation, L is the appropriately chosen Laplacian (or the Laplace-Beltrami operator on non-Euclidean domains). The solution to the diffusion equation is given by the heat operator H_t, so that $u_t = H_t(u_0) = \exp(-tL)u_0$. Importantly, the heat operator H_t is differentiable with respect to t, which was recently used in (Sharp et al., 2022) to use the diffusion equation with *a learnable time parameter* t as a way to replace convolution and enable long-range communication in the context of learning on curved surfaces.

As anticipated earlier, our key idea is to also use the diffusion process for information propagation. However, instead of using the heat operator as in (Sharp et al., 2022) we exploit *time-derivative diffusion* as a communication mechanism within graph neural networks. Taking the negative derivative of the heat operator with respect to t we obtain:

$$-\frac{\partial u_t}{\partial t} = Lu_t = T_t(u_0) = L\exp(-tL)u_0, \tag{6}$$

where $T_t(u) = LH_t(u)$ is the *time derivative diffusion* operator. We propose to use Eq. (6) to diffuse information between the nodes. Specifically, we construct a single layer of our model that we call TIDE, within a graph neural network as follows:

$$\mathcal{L}_k^{\text{TIDE}}(U) = \sigma\left(T_t(U)W^{(k)}\right) = \sigma\left(L\exp(-t_kL)UW^{(k)}\right). \tag{7}$$

Here k is the layer index, L is the Laplacian operator, and $W^{(k)}$ is the matrix of learnable weights associated with layer k.

Observe that our definition is similar to the standard message-passing layer defined in Eq. (2). However, crucially, our layer also includes the use of the diffusion operator $\exp(-t_kL)$ and in our resulting neural network architecture we make *both* the weight matrix W^k and the layer-wise time parameter t_k *learnable parameters*.

Our layer, as defined in Eq. (7) has two major properties:

1. First, by making the time t_k a learnable parameter, we allow the network to optimize the spatial extent of the diffusion and thus enable potentially global communication across graph nodes.
2. By using time-derivative diffusion instead of standard diffusion, we allow the network to revert to standard message-passing whenever necessary. Indeed, as shown below, our layer strictly generalizes the standard GCN layer, by simply setting $t = 0$. Moreover, since we start neural network training by initializing all learnable parameters (including the learnable time in Eq. (7)) around zero, the resulting network can optimize the spatial extent of its output, *only when necessary*.

We utilize the augmented normalized adjacency matrix from (Kipf & Welling, 2017) as the basis for diffusion in the following manner:

$$\tilde{L} := \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}. \tag{8}$$

Here $\tilde{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix with self-loops (binary or weighted) and D is the degree matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

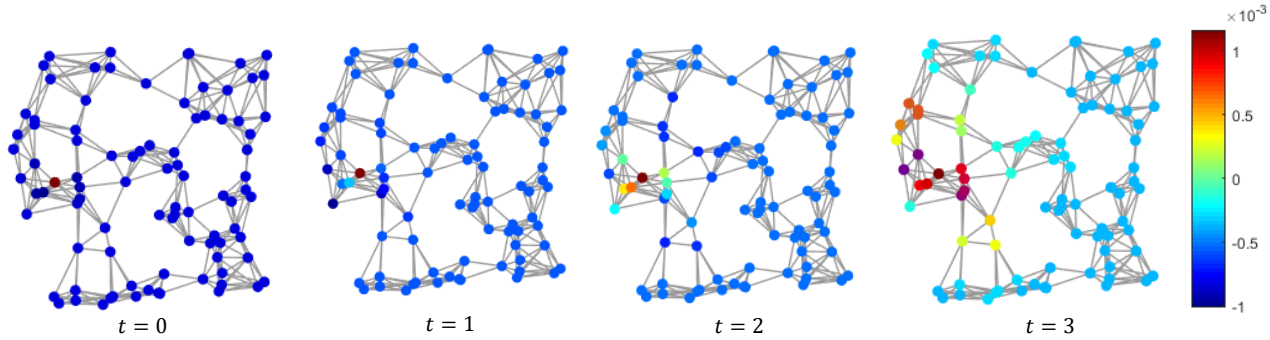

Figure 1. Influence of different values of the time parameter t in Eq. (7) on the spatial extent of the output of time-derivative diffusion. Note how for larger time values, the spatial support increases.

To provide intuition behind our approach, we illustrate in Fig. 1 the spatial extent of the output of time-derivative diffusion, and thus of our layer $\mathcal{L}^{\text{TIDE}}$ defined in Eq. (7) depending on the time parameter t. Observe that for small values of t, the support is local and the output of $\mathcal{L}^{\text{TIDE}}$ is concentrated at the center node. However, for larger time values, the spatial support increases, thus facilitating distant communication between nodes. Crucially, as mentioned above, in our framework, we let the time parameter t be a learnable variable, which allows the network to optimize it in a task-dependent manner.

Remark 1: *When the diffusion time t_k is set to 0, the diffusion block is equivalent to a GCN layer.* This follows directly from the properties of the operator exponential. Indeed, when $t_k = 0$, we know that $\exp(-t_k L)$ is simply the identity operator $\mathbf{I}$. Thus, at $t = 0$ we have $\mathcal{L}_k^{\text{TIDE}}(U) = \sigma\left(LUW^k\right) = \mathcal{L}_k(U)$ as defined in Eq. (2). which is one GCN layer, since we use the graph Laplacian operator L in both cases.

From this simple result, we can conclude that our network is at least as expressive as a GCN model. Moreover, the weights of a message-passing model, such as GCN, can be directly translated into the model based on Eq. (7), by using the same per-layer weight matrices and setting $t = 0$ in the latter. This simple flexibility allows our approach to increase the spatial support of every layer, only when it is useful for the underlying task.

Computation of the diffusion As computing the matrix exponential of large graph Laplacian matrices is both computationally expensive and numerically unstable (Moler & Van Loan, 2003), in our approach, we use spectral acceleration for the computation, as done in (Sharp et al., 2022). The key advantage of this formulation is that after a single pre-computation step, which calculates the Laplacian eigenbasis, the heat operator (and thus diffusion) for any time t can be calculated by elementwise exponentiation.

The eigenvector problem of the Laplacian can be formulated as follows: $L\phi_i = \lambda_i \phi_i$, where ϕ_i is the ith eigenvector of L and λ_i the corresponding eigenvalue sorted in ascending order by magnitude. After pre-computing the Laplacian eigenvectors and stacking the first l vectors as columns of the matrix $\mathbf{\Phi}$, the heat operator can then be obtained as follows:

$$H_t(u) = \mathbf{\Phi} \begin{bmatrix} e^{-t\lambda_0} \\ e^{-t\lambda_1} \\ \cdots \end{bmatrix} \odot \mathbf{\Phi}^\top u, \tag{9}$$

where $\mathbf{\Phi} = [\phi_i] \in \mathbb{R}^{V \times l}$ and $\odot$ denotes the Hadamard product (elementwise multiplication). In other words, we first project the signal u onto the basis given by $\mathbf{\Phi}$ via $u \to \mathbf{\Phi}^T u$. We then multiply (in an element-wise manner) each coefficient i by $\exp(-t\lambda_i)$ where λ_i is the eigenvalue corresponding to the ith eigenvector and then convert back to the standard basis by pre-multiplying by $\mathbf{\Phi}$.

By this low-rank basis projection of the operator, some information is lost. To compensate for information loss, we introduce the operator $P = \mathbf{I} - \mathbf{\Phi}\mathbf{\Phi}^T$. For any signal u, note that $H_0(u) + Pu = u$ holds. By incorporating the original signal mapped by P into the spectral approximation of the heat operator $H_t(u)$, we can efficiently compensate for lost information. The modified heat operator is defined as $\tilde{H}_t(u) = H_t(u) + \beta P u$. After simplifying and replacing the learnable scaling parameter $1 - \beta$ with α, the modified heat operator becomes:

$$\tilde{H}_t(u) = \mathbf{\Phi} \left(\begin{bmatrix} e^{-t\lambda_0} \\ e^{-t\lambda_1} \\ \cdots \end{bmatrix} \odot \alpha \mathbf{\Phi}^\top u \right) + \beta u \tag{10}$$

Note that Eq. (10) is differentiable with respect to t, which is essential in our setting, as t is learnable. We thus define $\tilde{T}_t(u) := L\tilde{H}_t(u)$ in Eq. (7) and make both α and β in Eq. (10) learnable. Note that this spectral approximation is

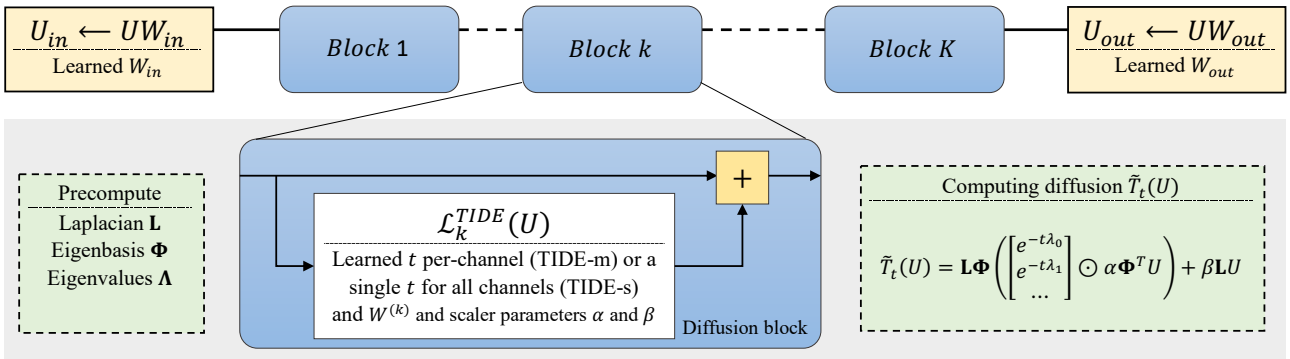

Figure 2. The framework of the proposed model. The network processes data from left to right. Our architecture is composed of several time derivative diffusion blocks for information propagation and aggregation, as well as of the input/output layers to convert to the appropriate input and output dimensions.

only one possibility for computing time derivative diffusion and in Section A.8 we show that the diffusion equation can also be implemented with the Euler method.

4.2. Time Diffusion Analysis

The main contribution of our model is learnable time diffusion. In the following, we briefly overview the relation between diffusion time and the size of the neighborhood in the information propagation through the graph. The diffusion Eq. (4) by the operator exponential $H_t = \exp(-tL)$ can be defined through its expansion as a Taylor series. For K-hop diffusion, this series can be truncated to bound the diffusion to a K- hop neighborhood as follows:

$$u_t^K = \sum_{k=0}^{K} \frac{(-t)^k}{k!} L^k u \tag{11}$$

Therefore the signal gets diffused only within the K-hop neighborhood. A simple bound on the difference between diffusion computed *purely* by looking at the K-hop neighborhood and diffusion computed over the entire graph can be obtained as:

$$\left\| u_t - u_t^K \right\| \leq \sum_{k=K+1}^{\infty} \frac{|-t|^k C^k}{k!} \tag{12}$$

where u is a normalized signal and C is the biggest eigenvalue of L (see also Appendix A.1).

Remark 2: *When considering the minimization of $\left\| u_t - u_t^K \right\|$ it is possible to minimize this term by either decreasing t or increasing K.*

Therefore, intuitively, one can think of the diffusion time t to correspond to the size of the neighborhood over which information is propagated. The smaller the time t, the smaller the neighborhood that can be used to to locally approximate diffusion over time t. Since in our architecture t is learned by the network, we interpret it as enabling an *optimizable receptive field* that can be adapted for different tasks and channels of the graph neural networks.

4.3. Architecture

In Fig. 2 we show a schematic view of the proposed TIDE network architecture. Our network architecture is composed of three main parts: one *Input layer*, one or more *Time Derivative Diffusion block(s)*, and one *Output layer*. A residual connection is adopted in each diffusion block to increase the accuracy and training performance, as shown in Appendix A.7. The network first processes each node feature individually with a linear layer. Subsequently, the diffusion unit processes the graph features. In each diffusion block, the network diffuses the node features of fixed D channels. In the TIDE-m model, each channel has its own learnable time parameter while in the TIDE-s model a learnable parameter t is common for all D channels. Finally, the output linear layer converts the learned output to the expected output dimensions.

5. Experiments

We compare our methods to strong baselines on typical node classification benchmarks and present novel long-distance communication experiments.

Setup For a fair comparison, we set similar values for the common hyperparameters in all baselines. For this purpose, we use 64 channels in the hidden layer and a dropout probability of 0.5. All models are trained with a maximum of 500 epochs with a learning rate of 0.01. In our approach we choose the highest validation accuracy to determine the number of diffusion blocks used. More implementation

Table 1. Comparison of the accuracy of proposed models on several benchmarks with baseline methods (mean±std).

Model	Cora	Citeseer	Pubmed	CoauthorCS	Computer	Photo	Ogbn-arxiv
GCN (Kipf & Welling, 2017)	83.30±0.36	68.23±0.91	76.78±0.31	90.17±0.50	81.01±0.65	91.71±0.67	65.91±0.12
GAT (Veličković et al., 2017)	81.83±0.42	69.19±0.53	75.49±0.43	90.15±0.35	80.25±0.52	91.57±0.41	54.23±0.22
GRAND (Chamberlain et al., 2021b)	80.71±0.86	68.06±0.18	74.61±0.25	90.59±0.21	72.96±0.49	84.17±0.34	59.29±0.12
GCNII (Chen et al., 2020)	79.94 ± 1.11	70.27±0.32	76.59±0.7	84.27±0.80	32.63±8.6	57.41±3.6	49.87±0.37
ACM (Luan et al., 2022)	81.83±0.12	69.03±0.02	73.3±0.63	**91.50±0.13**	77±0.65	**92.42±0.29**	66.23±0.42
DiffusionNet (Sharp et al., 2022)	80.96±0.50	70.00±0.91	73.09±0.15	89.52±0.22	74.72±0.66	87.17±0.26	54.79±0.16
TIDE-m	**84.47±0.43**	**70.32±0.68**	**77.59±0.04**	89.86±0.30	82.11±0.03	91.33±0.47	67.86±1.10
TIDE-s	84.31±0.36	70.24±0.80	77.24±0.62	90.21±0.12	**83.01±0.02**	92.06±0.51	**68.43±0.35**

details are described in Appendix A.2.

5.1. Node Classification

In the first experiment, we consider the standard node classification problem on a variety of benchmarks with different properties, described in Appendix A.3, and compare the performance of our proposed models, TIDE-m and TIDE-s, with several baselines. For the experimental setup we follow the methodology of (Chamberlain et al., 2021b).

Table 1 reports the mean accuracy and standard deviation of 10 different runs of our proposed models equipped with baselines. An ablation study for different numbers of blocks of the architecture is provided in Appendix A.4. The results obtained from most models are almost similar to their published numbers, except for GRAND (Chamberlain et al., 2021b) which only achieves the reported numbers after fine-tuning the hyperparameters. In addition, we also include a baseline "DiffusionNet", which shares a similar design to our approach but uses the standard heat diffusion like in (Sharp et al., 2022) in the diffusion block instead of our time-derivative diffusion.

As can be seen in Table 1, the TIDE models outperform baselines in most benchmarks. This comparison demonstrates the ability of the TIDE model to take advantage of global communication. We note that while TIDE-m has more learnable time parameters making the network more flexible, these additional parameters can lead to overfitting, making TIDE-s more accurate in some scenarios.

Most importantly, we observe a significant improvement compared to GCN (Kipf & Welling, 2017), which forms the basis of our approach. This demonstrates the effectiveness of incorporating time derivative diffusion as a communication mechanism and suggests the potential of applying the TIDE method in combination with other graph neural network techniques in future research.

5.2. Long Range Communication

As highlighted in Section 3, message-passing GNNs suffer from oversmoothing, typically when using more than 2-hop information propagation or more layers (Oono & Suzuki, 2019). In most cases, message-passing GNNs can apply two different layers for 2-hop neighbors without oversmoothing. Although this local information seems to be sufficient for some small-scaled citation graphs, for larger benchmarks long-distance communication can be useful. To demonstrate the effectiveness of long-range communication, we present two experiments on synthetic graphs.

In the first experiment, a synthetic graph for each benchmark is obtained by setting feature vectors of nodes that are not in the labeled training set to the zero vector. This scenario ensures that the final node labels are *only calculated* by using the information computed via the message passed from the labeled nodes and not inferred by the linear parts of the model. As seen in Table 7, Appendix A.5, the average distance between unlabeled and labeled nodes is long enough to enforce long-distance communication.

Table 2 shows the results of this experiment. This table further highlights the long-range communication capability of the proposed models, where TIDE models outperform the other GNNs in almost all datasets. Additional details are provided in Appendix A.5.

The second scenario is generated by the graphs introduced in (Karimi et al., 2018), which consists of 5000 nodes that are randomly partitioned into the train, validation, and test data with equal sizes. Inspired by the network with two groups and a homophily parameter h in this paper, we generate 10 different graphs by changing the h in the range 0.0 to 0.9, with interval 0.1. The parameter h indicates the likelihood of a node forming a connection to a neighbor with the same label. Therefore, as h increases, nodes prefer to connect with nodes of the same label, and thus with smaller h the long-range dependency communications will be prominent. The accuracies of all baselines against the homophily parameter are shown in Fig. 3. As shown in the figure, with a smaller h, long-range communication between nodes is more necessary. The proposed TIDE-m and TIDE-s models perform significantly better than the baselines with local message passing. Note also that although all models perform better as h increases, our approaches are the best-performing ones across *every* value of h. The Appendix A.6 includes additional experiments conducted

Table 2. Comparing different methods in the setting where feature vectors of unlabeled nodes are synthetically set to zero.

Model	Cora	Citeseer	Pubmed	CoauthorCS	Computer	Photo	Ogbn-arxiv	Average
GCN (Kipf & Welling, 2017)	57.87	40.16	41.21	46.93	59.82	74.09	56.65	53.82
GAT (Veličković et al., 2017)	56.19	40.73	**45.26**	**50.59**	53.14	66.74	42.47	50.73
GRAND (Chamberlain et al., 2021b)	52.79	41.45	40.83	24.71	17.58	24.39	55.58	36.76
DiffusionNet (Sharp et al., 2022)	60.05	60.08	42.56	25.4	19.28	30.12	35.77	39.04
TIDE-m	76.85	**61.45**	41.98	49.78	57.6	**78.49**	**57.75**	**60.56**
TIDE-s	**77.46**	60	41.95	40.90	**60.21**	77.4	57.27	59.31

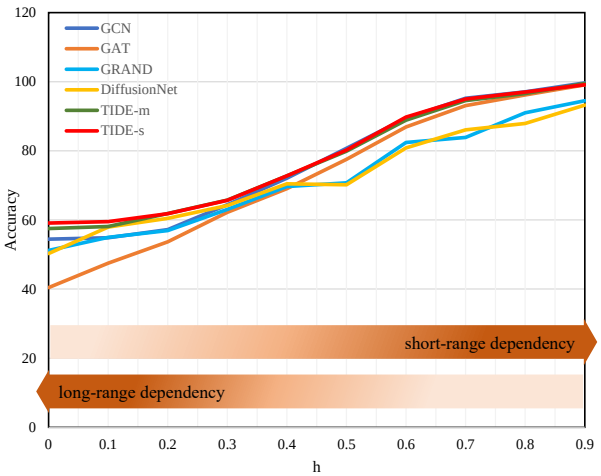

Figure 3. The accuracies of all baselines against the homophily parameter.

on graphs with different homophily rated.

5.3. Geometric Graphs

One weakness of traditional graph neural networks is the capability of node classification on geometric graphs. The authors of (Bouritsas et al., 2022) mention that typically graphs neural networks have problems operating on regular graphs, such as grids, triangle meshes, etc. To evaluate the performance of our approach on such data, we build a synthetic dataset based on the FAUST (Bogo et al., 2014) collection of shapes represented as triangle meshes. We convert the meshes into graphs, by simply using the graph structure of the triangle mesh (i.e., using vertices of the mesh as graph nodes and edges of the faces as graph edges). For the graph features, we will use the heat signature kernel (HKS) (Sun et al., 2009) with dimension 5. We develop three different synthetic settings: in **single** train and test nodes are on the same single graph,with a random 0.2, 0.3, 0.5 split for train, validation, and test, respectively. In the **multi** setting, we train the networks on the FAUST training set, which consists of 80 graphs, and tested on the FAUST test set, consisting of 20 test shapes. In the **mixed** setting, we train on the FAUST training set and tested on SHREC'07 four legged setting.

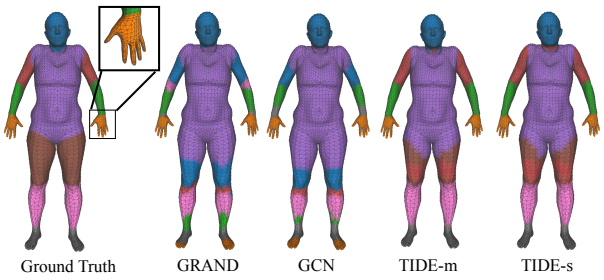

Figure 4. Qualitative example of labeling on one FAUST shape. Note in particular how the other algorithms completely fail to label the upper leg and upper arms.

Table 3. Testing the transfer capabilities of the graph networks on different datasets. **single**, **multi** and **mixed**.

Model	single	multi	mixed
GCN (Kipf & Welling, 2017)	69.21	65.90	65.83
GRAND (Chamberlain et al., 2021b)	78.46	69.54	46.32
TIDE-m	**94.11**	87.90	81.22
TIDE-s	91.73	**88.18**	**87.14**

We emphasize that for all baselines, we only use the graph structure of the shapes for learning and feature propagation, and ignore the node coordinates in 3D. As we can see in Table 3 our approach outperforms the baselines GCN and GRAND by a very significant margin in this scenario. Note also that we evaluate the generalization capabilities of our method, by evaluating meshes retrieved from the entirely different four legged SHREC'07 (Giorgi et al., 2007) dataset. Fig. 4 shows a qualitative result in the **single** scenario. We can see that for example, ours is the only method that is capable to recover the thigh as well as the upper arm.

5.4. Computation Cost

The computation of TIDE can be divided into three distinct parts: preprocessing (*pre*), training with derivative evaluation (*train*), and inference evaluation (*infer*). The preprocessing step includes the computation of Laplacian and eigendecomposition, which is executed only once on the CPU. For training and inference, TIDE utilizes standard linear algebra operations such as matrix multiplication and computation of the heat equation, which are efficiently executed on a GPU.

Table 4. Comparison of the runtime performance of TIDE and baseline methods on graphs of varying scales: Cora (2485 nodes), CoauthorCS (18333 nodes), and Ogbn-arxiv (169343 nodes). The reported runtime values are in seconds.

Model		Small (Cora)	Medium (CoauthorCS)	Large (Ogbn-arxiv)
GCN	*pre*	0.5966	5.5038	213.79
	train	0.1217	0.1488	0.2411
	inference	0.0452	0.0736	0.107
GAT	*pre*	–	–	–
	train	0.0345	0.039	0.0921
	inference	0.024	0.0256	0.0414
GRAND	*pre*	0.5492	5.9851	209.19
	train	1.5322	1.8333	2.9816
	inference	0.9646	0.5536	1.5923
DiffusionNet	*pre**	0.5650	5.5306	214.19
	train	0.0974	0.1131	0.1437
	inference	0.0375	0.0554	0.0679
TIDE-m	*pre**	0.5650	5.5306	214.19
	train	0.1188	0.1656	0.2551
	inference	0.049	0.0726	0.1099
TIDE-s	*pre**	0.5650	5.5306	214.19
	train	0.0975	0.1649	0.2344
	inference	0.0415	0.0747	0.0996

* Pre-processing times are the same for these approaches.

We conducted experiments to evaluate the runtime performance of TIDE and compared it with several baseline methods on graphs of varying sizes, including small, medium, and large-scale graphs. The results of these experiments are summarized in Table 4.

According to the results presented in Table 4, the TIDE model performs comparably to spectral methods such as GCN and DiffusionNet in terms of runtime performance, while significantly outperforming the GRAND model. Nevertheless, the runtime performance achieved by the GAT models outperforms all the methods since it allows the model to selectively attend to relevant nodes in the graph and also shares parameters across all nodes. These results highlight the efficiency of TIDE, which achieves runtime performance comparable to other state-of-the-art methods.

5.5. Ablation Studies

We present ablation studies for Residual Connection and Diffusion Time in appendices A.7 and A.9, respectively.

6. Conclusion, Limitations & Future work

In this work, we introduced a novel neural network architecture for graph learning. Our key idea is to use time derivative with a learnable time parameter to augment the message-passing component of graph neural networks and enable long-range communication efficiently. Our method is similar in efficiency to the strong GCN baseline, and scales well to large problem sizes, as no parameters depend on the number of nodes, and no expensive integration needs to be calculated during training.

As we build TIDE upon the standard message-passing paradigm, our approach is well-situated within the Weisfeiler-Lehman 1 category. As such, our current method cannot distinguish certain non-isomorphic graphs outside of this category. Nevertheless, we believe that the idea of time-derivative diffusion can be incorporated into other frameworks, such as recent methods with WL-3 expressive power. Moreover, it will also be interesting to extend our method to *anisotropic diffusion* for information communication, while maintaining efficiency and differentiability of the time parameter. We leave this as an exciting direction for future work.

7. Acknowledgements

The authors acknowledge the anonymous reviewers for their valuable suggestions. Parts of this work were supported by the ERC Starting Grant No. 758800 (EXPROTEA) and the ANR AI Chair AIGRETTE.

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Avelar, P. H., Tavares, A. R., Gori, M., and Lamb, L. C. Discrete and continuous deep residual learning over graphs. *arXiv preprint arXiv:1911.09554*, 2019.

Behmanesh, M., Adibi, P., Ehsani, S. M. S., and Chanussot, J. Geometric multimodal deep learning with multiscaled graph wavelet convolutional network. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022. doi: 10.1109/TNNLS.2022.3213589.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Bogo, F., Romero, J., Loper, M., and Black, M. J. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA, June 2014. IEEE.

Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Chamberlain, B., Rowbottom, J., Eynard, D., Di Giovanni, F., Dong, X., and Bronstein, M. Beltrami flow and neural diffusion on graphs. *Advances in Neural Information Processing Systems*, 34:1594–1609, 2021a.

Chamberlain, B., Rowbottom, J., Gorinova, M. I., Bronstein, M., Webb, S., and Rossi, E. GRAND: graph neural diffusion. In *International Conference on Machine Learning*, pp. 1407–1418. PMLR, 2021b.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *ICML*, pp. 1725–1735, 2020.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural odes. *Advances in Neural Information Processing Systems*, 32, 2019.

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. How to train your neural ode: the world of jacobian and kinetic regularization. In *International conference on machine learning*, pp. 3154–3164. PMLR, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Giorgi, D., Biasotti, S., and Paraboschi, L. Shape retrieval contest 2007: Watertight models track. *SHREC competition*, 8(7):7, 2007.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 1025–1035. Curran Associates Inc., 2017. ISBN 9781510860964.

Hou, T. and Qin, H. Continuous and discrete mexican hat wavelet transforms on manifolds. *Graphical Models*, 74 (4):221–232, 2012.

Karimi, F., Génois, M., Wagner, C., Singer, P., and Strohmaier, M. Homophily influences ranking of minorities in social networks. *Scientific Reports*, 8:1–12, 2018. ISSN 2045-2322. doi: https://doi.org/10.1038/s41598-018-29405-7.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017.

Kirgo, M., Melzi, S., Patane, G., Rodola, E., and Ovsjanikov, M. Wavelet-based heat kernel derivatives: Towards informative localized shape analysis. In *Computer Graphics Forum*, volume 40, pp. 165–179. Wiley Online Library, 2021.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Li, G., Muller, M., Thabet, A., and Ghanem, B. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9267–9276, 2019a.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018a.

Li, R., Wang, S., Zhu, F., and Huang, J. Adaptive graph convolutional neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018b.

Li, X., Wong, T.-K. L., Chen, R. T., and Duvenaud, D. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pp. 3870–3882. PMLR, 2020.

Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pp. 3835–3845. PMLR, 2019b.

Lim, D., Hohne, F. M., Li, X., Huang, S. L., Gupta, V., Bhalerao, O. P., and Lim, S.-N. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *Advances in Neural Information Processing Systems*, 2021.

Liu, M., Gao, H., and Ji, S. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020.

Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W., and Precup, D. Revisiting heterophily for graph neural networks. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.

Moler, C. and Van Loan, C. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Poli, M., Massaroli, S., Park, J., Yamashita, A., Asama, H., and Park, J. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

Sarlin, P.-E., DeTone, D., Malisiewicz, T., and Rabinovich, A. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4938–4947, 2020.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Sharp, N., Attaiki, S., Crane, K., and Ovsjanikov, M. Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG)*, 41(3):1–16, 2022.

Sun, J., Ovsjanikov, M., and Guibas, L. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pp. 1383–1392. Wiley Online Library, 2009.

Thorpe, M., Nguyen, T. M., Xia, H., Strohmer, T., Bertozzi, A., Osher, S., and Wang, B. Grand++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2021.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wieder, O., Kohlbacher, S., Kuenemann, M., Garon, A., Ducrot, P., Seidel, T., and Langer, T. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12, 2020.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Xhonneux, L.-P., Qu, M., and Tang, J. Continuous graph neural networks. In *International Conference on Machine Learning*, pp. 10432–10441. PMLR, 2020.

Xu, B., Shen, H., Cao, Q., Qiu, Y., and Cheng, X. Graph wavelet neural network. In *International Conference on Learning Representations*, 2019.

Yao, L., Mao, C., and Luo, Y. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 7370–7377, 2019.

Zhang, X., Liu, H., Li, Q., and Wu, X.-M. Attributed graph clustering via adaptive graph convolution. *arXiv preprint arXiv:1906.01210*, 2019.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Zhuang, C. and Ma, Q. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, pp. 499–508, 2018.

A. Appendix

A.1. Threshold for Diffusion

The Taylor series for the diffusion equation is defined by:

$$u_t = e^{-tL}u = \sum_{k=0}^{\infty} \frac{1}{k!}(-tL)^k u = \sum_{k=0}^{\infty} \frac{-t^k}{k!} L^k u \tag{13}$$

Intuitively, for any $t \neq 0$, this series propagates information from the whole graph, as no factor in front of the power of the Laplacians is 0.
Eq. (12) can be derived simply as follows:

$$\left\| u_t - u_t^K \right\| = \left\| \sum_{k=K+1}^{\infty} \frac{(-tL)^k u}{k!} \right\| \leq \sum_{k=K+1}^{\infty} \frac{|-t|^k C^k}{k!} \tag{14}$$

A.2. Implementation Details

The models are implemented in PyTorch, and the torch geometric library is also incorporated in addition to the standard PyTorch. To accelerate learning, GPU acceleration is utilized, while the diffusion operator and the gradient operator are preprocessed on a CPU using the SciPy library. The experiments are conducted on an NVIDIA A100 GPU with 40 GB of GPU memory. Despite the fact that the model has a small number of parameters, it can be trained on any GPU.

A.3. Datasets Properties

The properties of different graph datasets used in the experiments are summarized in Table 5.

Table 5. Datasets properties

Graph	#Nodes	#Edges	#Node featues	#Class	Avg. node deg.	Graph diameter	Label rate
Cora	2485	5069	1433	7	4.07	1.53	0.056
Citeseer	2120	3679	3703	6	3.47	1.44	0.057
PubMed	19717	44324	500	3	4.49	18	0.003
CoauthorCS	18333	81894	6805	15	8.93	24	0.016
Computers	13381	245778	767	10	36.73	0.16	0.015
Photos	7487	119043	745	8	31.79	0.23	0.021
Ogbn-arxiv	169343	1166243	128	40	13.67	23	1

A.4. Number of Blocks and Oversmoothing

We assess the effectiveness of our proposed models by examining the performance across different numbers of blocks present in the architecture. Fig. 5 illustrates that the network accuracy is influenced by the dataset when the number of blocks in the network is limited to 3 or fewer. Furthermore, we provide in Table 6 an analysis of how the test and validation accuracies vary as the number of diffusion blocks increases. The table illustrates that the TIDE-m model performs relatively well with up to 16 layers, and only begins to exhibit oversmoothing signs with 32 layers. We observe that our method maintains its high performance even with an increase in the number of diffusion blocks. Furthermore, our approach appears to be less prone to oversmoothing when compared to GCN.

A.5. Long-range Dependency

To demonstrate the capability of our model in taking advantage of long-range communication, we present further insights into the scenario of zeroing out the test features. To perform this experiment, we followed the data distribution used in the node classification experiment to determine the train, validation, and test data. In Table 7, we summarize the average and max distances between unlabeled nodes and the closest labeled one of each graph. According to these statistics in Table 7, the mean distance in each graph is quite far, which led to the fact that information aggregation is not possible by simply averaging from the neighbors. Therefore, long-range communication assumes paramount importance in this

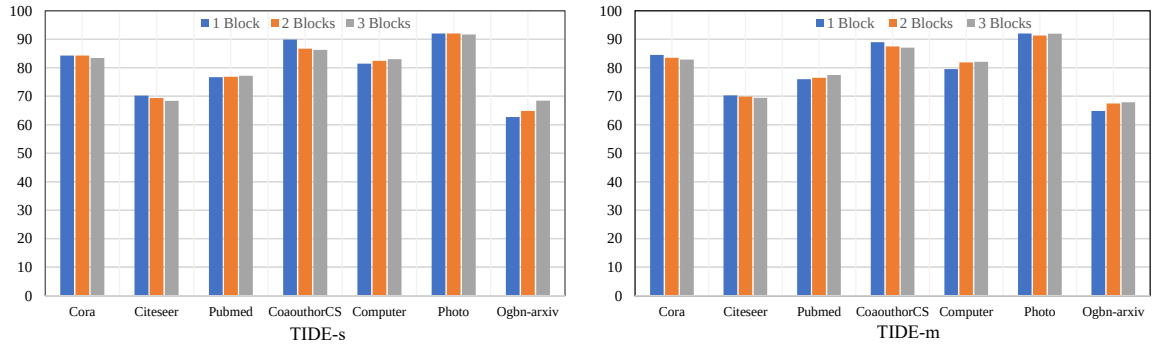

Figure 5. Performances of the proposed models (TIDE-s and TIDE-m) by choosing the different number of blocks.

Table 6. Performance of TIDE-m on the validation and test subset with $1, 2, 4, ..., 32$ diffusion blocks

Dataset	Model	2^0	2^3	2^4	2^5
Cora	TIDE-m (*test*)	**84.47±0.0**	**78.63±0.36**	**78.38±0.86**	**40.61±8.2**
	TIDE-m (*val*)	82.49x0.41	78.74±1.1	78.08±1.1	40.55±9.2
	GCN (*test*)	72.57±0.23	74.03±0.42	57.11±0.78	27.78±0.44
	GCN (*val*)	74.51±0.39	75.04±0.19	55.32±0.24	32.99±0.89
Citeseer	TIDE-m (*test*)	**70.48±0.23**	**66.85±0.34**	**65.73±0.57**	**55.73±4.9**
	TIDE-m (*val*)	71.97±0.2	67.30±0.36	66.47±0.0	52.20±5.2
	GCN (*test*)	61.05±0.43	57.23±0.34	56.55±0.23	22.43±0.81
	GCN (*val*)	63.23±0.81	58.01±32	58.62±0.92	19.25±0.54

particular scenario. As shown in Table 7, the average distance between unlabeled nodes and labeled ones is more than 1 on all datasets, and even on Citeseer, the average distance is 2.91 which this value intelligibly reveals the reason for the significant improvement of TIDE compare with GCN in Table 2. Therefore, the 1-hop neighborhood of nodes becomes ineffective since it yields a vector of zeros. The table demonstrates that for Photo, Computer, and Ogbn-arxiv the mean distance is smaller than a 2-hop neighborhood. In these metrics, although the accuracy of TIDE is better than other baselines, we would expect a smaller accuracy increment, which is indeed the case. In addition, Table 2 shows that with the bigger mean and max distance, TIDE outperforms previous methods. It is worth noting that the limited percentage of labeled data in Pubmed (0.29%) leads to a strong possibility of overfitting, and making conclusions difficult to interpret. As a consequence, we can conclude that the ability of the TIDE model to communicate over long-distances is evidenced by the 0-node label experiment.

Table 7. Statistics of graphs with zeroing out the test feature. These statistics are the average and max distances between each unlabeled node and its closest labeled one on each graph. They reflect the long-range dependency among nodes in this scenario.

Graph	Average distance	Max distance
Cora	2.39	9
Citeseer	2.91	13
Pubmed	3.54	9
CoauthorCS	2.41	7
Computer	1.79	5
Photo	1.55	5
Ogbn-arxiv	1.63	10

A.6. Experiments on Graphs with Different Homophily Rates

In order to conduct a more comprehensive analysis, we investigate the performance of our proposed model on a series of heterophilic graphs, introduced in (Lim et al., 2021). These graphs exhibit varying levels of homophily and sizes, which are delineated in Table 8. The homophily rate h denotes the degree to which nodes in the graph connect with similar nodes (homophily) versus nodes with dissimilar nodes (heterophily).

Table 9 indicates that the proposed model achieved the best accuracy in six out of eight graphs. By comparing the results of the proposed model with the baselines, we can conclude that the proposed model outperforms the baselines on heterophilic graphs. Specifically, for larger graphs such as Snap-patents with a higher heterophily ratio, the proposed models achieved significantly higher performance scores than the baselines. This suggests that the proposed model is more effective in capturing the long dependency in heterophilic graphs, which is an important finding that can inform the development of more effective models for real-world scenarios.

Table 8. Dataset properties. The parameter $h[0, 1]$, represents the edge homophily ratio. When $h \to 1$ the graph exhibits high levels of homophily, whereas $h \to 0$, the graph displays strong levels of heterophily.

Graph	#Nodes	#Edges	#Node featues	#Class	Class type	h
Chameleon	2,277	36,101	2,325	5	Wiki pages	0.23
Actor	7,600	29,926	931	5	Actors in movies	0.22
Cornell	183	295	1,703	5	Web pages	0.3
Texas	183	309	1,703	5	Web pages	0.11
Wisconsin	251	499	1,703	5	Web pages	0.21
Genius	421,961	984,979	12	2	marked act.	0.618
Twitch-gamers	168,114	6,797,557	7	2	mature content	0.545
Snap-patents	2,923,922	13,975,788	269	5	time granted	0.073

Table 9. Comparison of the accuracy of proposed models on heterophilic graphs with baseline methods (mean±std).

Model	Chameleon	Actor	Cornell	Texas	Wisconsin	Genius	Twitch-gamer	Snap-patents
GCN	45.18±0.62	29.38±0.5	43.24±1.3	63.51±1.9	54.92±9.7	80.87±0.13	60.60±0.19	36.84±0.37
GAT	44.96±6.2	28.88±1.0	54.05±1.1	62.16±0.08	55.88±1.4	79.83±0.23	53.08±0.16	38.76±0.75
GRAND	50.33±0.47	35.00±0.28	55.41±1.9	**67.62±1.9**	64.86±1.3	82.47±0.08	59.85±0.03	38.89±0.42
DiffusionNet	**53.84±1.1**	34.44±0.33	56.76±0.6	62.16±0.0	62.78±2.8	82.59±0.12	55.72±1.6	30.69±0.014
TIDE-m	52.08±1.1	36.18±0.47	58.11±1.9	64.86±1.5	**69.61±1.4**	**83.03±0.06**	**60.81±0.04**	40.56±1.7
TIDE-s	51.75±0.47	**36.64±0.47**	**59.46±1.9**	63.81±1.9	68.63±1.4	83.01±0.06	60.40±0.13	**40.75±0.58**

A.7. Ablation about Residual Connection

Table 10 presents the results of the ablation study conducted on the residual connection. The findings indicate that the combination of residual connection and diffusion is the most effective network architecture. Additionally, it is noteworthy that utilizing solely the residual connection without the diffusion model is significantly less effective.

A.8. Direct Implicit Timestep

Matrix exponential approximations in diffusion We show that our method does not depend on the type of diffusion approximation used. In particular, we compare the spectral approximation advocated in Section 4.1 with using the *implicit Euler* scheme to simulate diffusion (Sharp et al., 2022).

Since the graph Laplacians can get fairly big, we only mention the results of the small datasets, Cora and Citeseer as shown in Table 11. Observe that we obtain worse results using implicit Euler diffusion approximation, compared to when using spectral approximation. In addition, the latter benefits from the operator P, which recovers some of the information. The first approach is mathematically exact. Hence we did not introduce any signal recovery mechanism here. However, with the huge matrices and possibility of singularities, the method is numerically less stable and results in worse results. In addition, the spectral approximation method is more scalable to large graphs, which is why we adopt it throughout our work.

Instead of using the spectral acceleration to compute the time derivative diffusion, we can also use the implicit Euler method as follows:

$$H_t(u) := L(I + tL)^{-1}u. \tag{15}$$

It requires solving a sparse linear equation system for each iteration during training and testing. The implicit version makes this approach numerically stable compared to the direct Euler version. In PyTorch, it is possible to solve those systems and

Table 10. Comparison of the effect of the residual connection on several benchmarks (mean±std)

Model	Cora	Citeseer	Pubmed	CoauthorCS	Computer	Photo	Ogbn-arxiv
TIDE-m	**84.47±0.43**	**70.32±0.68**	77.59±0.04	87.07±2.10	82.11±0.03	91.33±0.47	67.86±1.10
without residual	82.44±0.57	68.79±0.57	**77.76±0.33**	88.47±0.15	82.01±1.2	91.76±0.11	58.80±9.50
TIDE-s	84.31±0.36	70.24±0.80	77.24±0.62	86.29±5.90	**83.01±0.02**	**92.06±0.51**	**68.43±0.35**
without residual	83.71±0.79	68.39±0.23	77.37±0.76	**88.73±0.06**	82.46±0.26	91.87±0.52	64.60±0.67
without diffusion	54.89±0.00	55.59±0.01	66.20±0.00	86.47±0.00	61.61±0.00	77.02±0.00	56.60±0.70

back-propagate through them. However, on graph datasets, the Laplacian matrix can be rather large, and it is only with CPU memory possible to solve this equation system. Hence, its training speed is drastically reduced.

In addition, we also found that only using the low-frequency approximation of diffusion is beneficial to regularize network training.

Table 11. Using Spectral and implicit dense methods to calculate the diffusion equation

Method	Cora	Citeseer
Spectral	**84.47±0.43**	**70.32±0.68**
implicit dense	82.18±0.00	68.60±0.01

A.9. Diversity of Learned Time t and Channel Analysis

The novel learning parameter of the TIDE model is time, which controls the diffusivity of the information of the node features. Since our model tries to find optimal times for each channel, to show how its performance can change depending on this parameter, we explore a range of $[0, 2]$ with a step size of 0.1 to identify the optimal times for each channel. In Fig. 6, we compare the accuracy of the TIDE-s model on four datasets at different fixed time values t. This experiment aims to evaluate how the choice of diffusion time affects the performance of the model and to determine the optimal value of t for this particular dataset. We can see in this figure, for diffusion time $t = 0$ the accuracy of our model and GCN are approximately equivalent. However, when t is larger than zero, the best accuracy will be obtained with a specific value of time $t > 0$ on each benchmark.

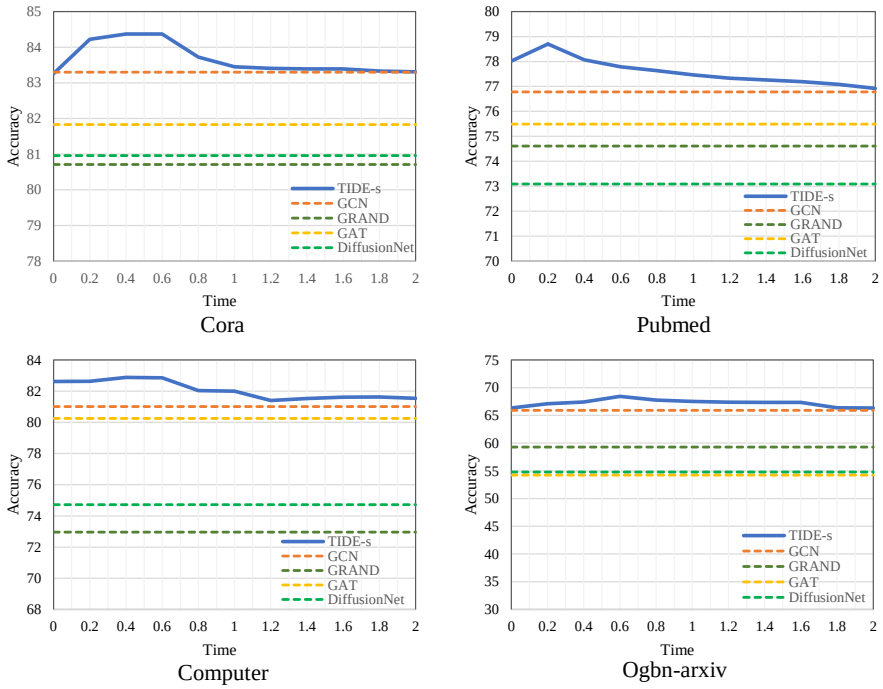

Figure 6. Comparison of accuracy of four benchmarks by changing the time parameter.

In Fig. 7 we can see the behavior of the learned time of the TIDE-m model on several sampled channels during the training on Computer, Photo, and Ogbn-arxiv datasets. It is observed that the learned time starts from a certain initial value and then gradually converges to an optimal value after an initial learning period.

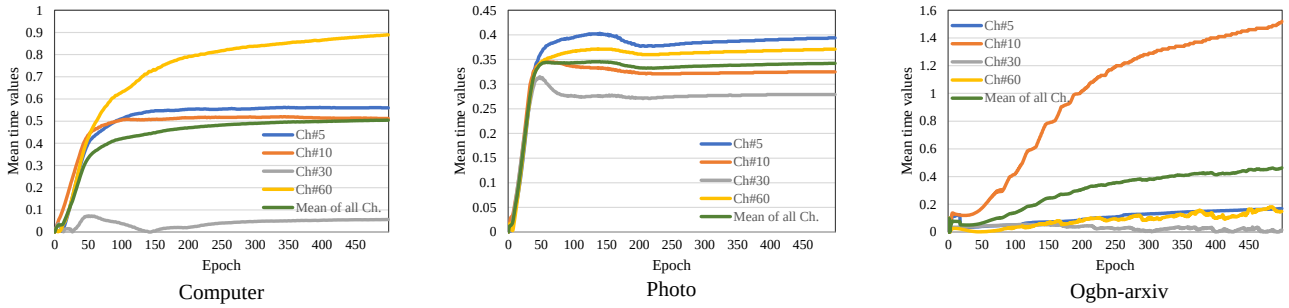

Figure 7. Value of time learned on channels 5, 10, 30, and 60 during the training. The green curve indicates the mean time values of these channels.