Bundle Neural Networks for message diffusion on graphs

Jacob Bamberger [1] **Federico Barbero** [1] **Xiaowen Dong** [1] **Michael Bronstein** [1]

Abstract

The dominant paradigm for learning on graph-structured data is message passing. Despite being a strong inductive bias, the local message passing mechanism suffers from pathological issues such as over-smoothing, over-squashing, and limited node-level expressivity. To address these limitations we propose Bundle Neural Networks (BuNN), a new type of GNN that operates via *message diffusion* over *flat vector bundles* – structures analogous to connections on Riemannian manifolds that augment the graph by assigning to each node a vector space and an orthogonal map. We show that BuNNs can mitigate over-smoothing and over-squashing, and that they are universal compact uniform approximators on graphs. We showcase the strong empirical performance of BuNNs over real-world tasks, achieving state-of-the-art results on several standard benchmarks.

1. Introduction

Graph Neural Networks (GNNs) (Sperduti, 1993; Scarselli et al., 2009) are widely adopted machine learning models designed to operate over graph structures. A majority of GNN models function locally, respecting the topology by having nodes exchange messages directly with neighboring nodes. Such models are referred to as *Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017)*. While message passing has repeatedly proven to be a useful inductive bias in many tasks, the local mechanism was shown to suffer from certain pathological issues, namely over-smoothing (Li et al., 2018; Oono & Suzuki, 2020; Cai & Wang, 2020), over-squashing (Alon & Yahav, 2021; Topping et al., 2022; Di Giovanni et al., 2023), and limited expressivity (Xu et al., 2019; Morris et al., 2019).

[1] University of Oxford. Correspondence to: Jacob Bamberger <jacob.bamberger@cs.ox.ac.uk>.

Presented at the Geometry-grounded Representation Learning and Generative Modeling Workshop at the 41st International Conference on Machine Learning, Vienna, Austria. Copyright 2024 by the author(s).

To address these issues, we propose **Bundle Neural Networks (BuNNs)**, a new type of GNN that operates over *flat vector bundles* – a structure that augments the graph by assigning to each node a vector space and an orthogonal map. BuNNs *do not perform 'explicit' message passing* through multiple steps of information exchange between neighboring nodes, but instead operate via what we call **message diffusion**. Each layer involves a node update step, and a diffusion step evolving the features according to a vector diffusion PDE as in Singer & Wu (2012). The resulting architecture enjoys the desirable properties of Sheaf Neural Networks (Bodnar et al., 2022), in that they can avoid over-smoothing, but are more scalable and achieve better performance on a range of benchmark datasets. We also show how the continuous nature of message diffusion together with the bundle structure allows to mitigate issues such as over-squashing.

In summary, our **contributions** are the following:

- We derive BuNNs from heat equations over flat vector bundles, which are more amenable to computation than general vector bundles (Section 3).
- We design synthetic experiments to show that BuNNs can mitigate over-smoothing and over-squashing (Section 4.1). We give formal proofs of such claims in the Appendix (Section C.)
- We show that BuNNs perform well on a range of tasks, for instance achieving a new state-of-the-art result on the `Peptides-func` dataset (Section 4.2).

2. Background

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be an undirected graph on $n = |\mathsf{V}|$ nodes with edges E. We represent the edges via an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ where the entry $\mathbf{A}_{uv}$ for $u, v \in \mathsf{V}$ is 1 if the edge $(u, v) \in \mathsf{E}$ and 0 otherwise. Let $\mathbf{D}$ be the diagonal degree matrix with entry $\mathbf{D}_{vv} = d_v$ equal to the degree of v. The graph Laplacian is defined as $\mathbf{L} := \mathbf{D} - \mathbf{A}$ and the random walk normalized graph Laplacian is defined as $\mathcal{L} := \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$. We assume that at each node $v \in \mathsf{V}$ we are given a c-dimensional signal $\mathbf{x}_v$ and group such signals into a matrix $\mathbf{X} \in \mathbb{R}^{n\times c}$. A *feature transformation* on a graph G is a permutation-equivariant map $f_{\mathsf{G}} : \mathbb{R}^{n\times c} \to \mathbb{R}^{n\times c'}$ that transforms the node signals.

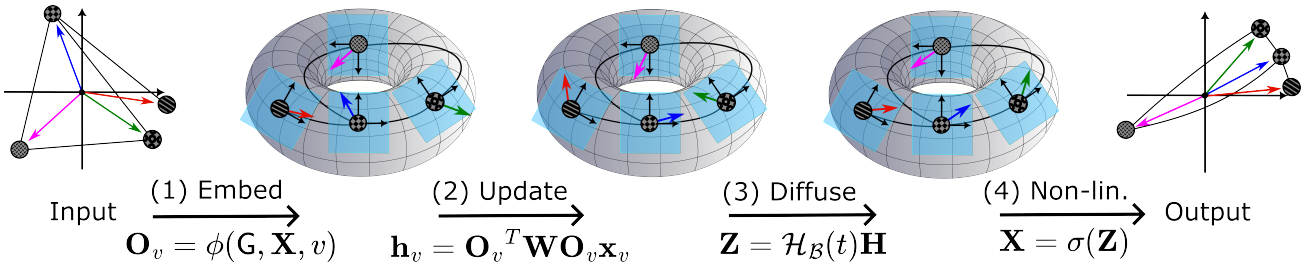

Figure 1. Example of the message diffusion framework on a graph with 4 nodes and 4 edges. From left to right: At first each node is represented in global coordinates. Next, each node gets a local reference frame which is aligned to an underlying manifold structure (represented as a torus for visual aid). Third, the feature vectors are diffused according to the heat equation (in this figure vectors are diffused until the stable state is achieved at $t = \infty$ where the features are aligned according to their local coordinates). Finally, the resulting embedding are represented in global coordinates.

Cellular sheaves. A *cellular sheaf* (Curry, 2014) $(\mathcal{F}, \mathsf{G})$ over an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ augments G by attaching to each node v and edge e a vector space space called *stalks* and denoted by $\mathcal{F}(v)$ and $\mathcal{F}(e)$, usually the stalks are copies of $\mathbb{R}^d$ for some d. Additionally, every incident node-edge pair $v \trianglelefteq e$ gets assigned a linear map between stalks called *restriction maps* and denoted $\mathcal{F}_{v \trianglelefteq e} : \mathcal{F}(v) \to \mathcal{F}(e)$. Given two nodes v and u connected by an edge $(v,\ u)$, we can *transport* a vector $\mathbf{x}_v \in \mathcal{F}(v)$ from v to u by first mapping it to the stalk at $e = (v, u)$ using $\mathcal{F}_{v \trianglelefteq e}$, and mapping it to $\mathcal{F}(u)$ using the transpose $\mathcal{F}^T_{u \trianglelefteq e}$. As a generalization of the graph adjacency matrix, the *sheaf adjacency matrix* $\mathbf{A}_{\mathcal{F}} \in \mathbb{R}^{nd \times nd}$ is defined as a block matrix in which each $d \times d$ block $(\mathbf{A}_{\mathcal{F}})_{uv}$ is $\mathcal{F}^T_{u \trianglelefteq e}\mathcal{F}_{v \trianglelefteq e}$ if there is an edge between u and v and $\mathbf{0}_{d \times d}$ if there is not. We similarly define the block diagonal *degree matrix* $\mathbf{D}_{\mathcal{F}} \in \mathbb{R}^{nd \times nd}$ as $(\mathbf{D}_{\mathcal{F}})_{vv} := d_v \mathbf{I}_{d \times d}$. Finally, we define the sheaf Laplacian as $\mathbf{L}_{\mathcal{F}} = \mathbf{D}_{\mathcal{F}} - \mathbf{A}_{\mathcal{F}}$. These matrices act as bundle generalizations of their well-known standard graph counterparts and we recover such matrices when $\mathcal{F}(v) \cong \mathbb{R}$ and $\mathcal{F}_{v \trianglelefteq e} v = 1$ for all $v \in \mathsf{V}$ and $e \in \mathsf{E}$.

Vector bundles. When restriction maps are orthogonal, we call the sheaf a *vector bundle*, a structure analogous to connections on Riemannian manifolds. For this reason, the sheaf Laplacian also takes the name *connection Laplacian* (Singer & Wu, 2012). In this case the product $\mathcal{F}^T_{u \trianglelefteq e}\mathcal{F}_{v \trianglelefteq e}$ is also orthogonal and is denoted $\mathbf{O}_{uv}$ referring to the transformation a vector undergoes when moved across a manifold via parallel transport. In this case we denote the node-stalk at v by $\mathcal{B}(v)$, the bundle-adjacency by $\mathbf{A}_{\mathcal{B}}$ and the bundle Laplacian $\mathbf{L}_{\mathcal{B}}$, and its random-walk normalized version $\mathcal{L}_{\mathcal{B}} := \mathbf{I}_{dn \times dn} - \mathbf{D}_{\mathcal{B}}^{-1}\mathbf{A}_{\mathcal{B}}$. We illustrate the difference between the bundle Laplacian and standard graph Laplacian in Figure 1.

Consider a d-dimensional vector field over the graph, i.e. a d-dimensional feature vector at each node denoted and stacked column-wise into a vector $\mathbf{X} \in \mathbb{R}^{nd}$. Similarly to the graph case, the operation $\mathbf{D}_{\mathcal{B}}^{-1}\mathbf{A}_{\mathcal{B}}\mathbf{X}$ is an averaging over the vector field, and $\mathcal{L}_{\mathcal{B}}$ a measure of smoothness, since:

$$\left(\mathbf{D}_{\mathcal{B}}^{-1}\mathbf{A}_{\mathcal{B}}\mathbf{X}\right)_u = \frac{1}{d_u} \sum_{u:(v,u)\in\mathsf{E}} \mathbf{O}_{uv}\mathbf{x}_v \ \in \mathbb{R}^d, \text{ and}$$

$$\left(\mathcal{L}_{\mathcal{B}}\mathbf{X}\right)_u = \frac{1}{d_u} \sum_{u:(v,u)\in\mathsf{E}} \left(\mathbf{x}_u - \mathbf{O}_{uv}\mathbf{x}_v\right) \ \in \mathbb{R}^d.$$

3. Bundle Neural Networks

In this Section, we derive the Bundle Neural Network model from heat diffusion equations over flat vector bundles. We can write a *heat diffusion equation* (or simply *heat equation*) over a vector bundle as a differential equation whose evolution equation with initial condition $\mathbf{X}(0) = \mathbf{X}$ is:

$$\partial_t \mathbf{X}(t) = -\mathcal{L}_{\mathcal{B}}\mathbf{X}(t)\ ,\ \mathbf{X}(0) := \mathbf{X}\ .$$

It is well known (see Hansen & Gebhart (2020)) that the solution can be written using matrix exponentiation as:

$$\mathbf{X}(t) = \exp(-t\mathcal{L}_{\mathcal{B}})\mathbf{X}(0)$$

We call the operator $\mathcal{H}_{\mathcal{B}}(t) := \exp(-t\mathcal{L}_{\mathcal{B}}) \in \mathbb{R}^{nd \times nd}$ the *bundle heat kernel*. Computing the heat kernel is necessary to solve the heat equation. An exact solution can be computed using spectral methods. For small t, one can instead consider the truncated Taylor expansion centered at 0. Bodnar et al. (2022) instead approximates the solution using the Euler discretization with unit time step of the heat equation.

However, all these methods pose a challenge when the bundle is learned as in Bodnar et al. (2022), since the heat kernel has to be recomputed after every gradient update of the learned bundle structure. This high computational overhead limits the usability of Sheaf Neural Networks in applications.

Flat vector bundles. To address the scalability issues of general vector bundles, we consider the special case of *flat vector bundles* where every node u gets assigned an orthogonal map $\mathbf{O}_u$, and restrictions factorize as $\mathbf{O}_{vu} = \mathbf{O}_v^T\mathbf{O}_u$.

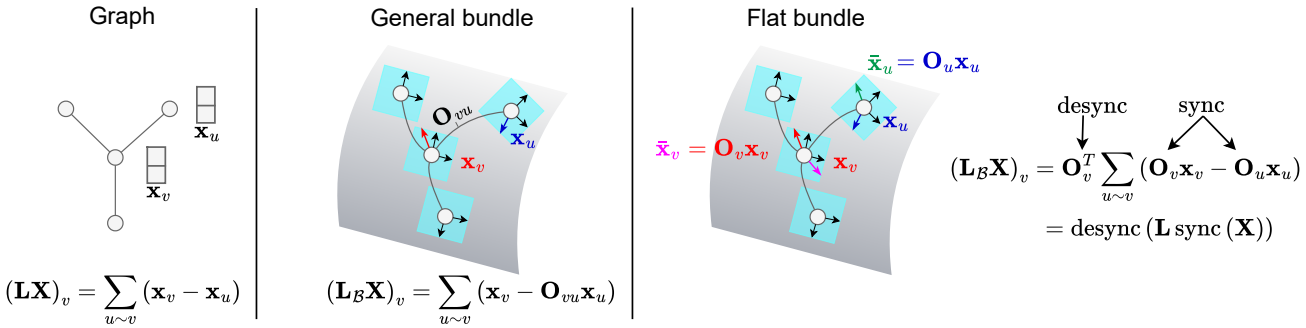

Figure 2. Comparison of different Laplacians, from left to right: standard Laplacian, general bundle Laplacian, flat bundle Laplacian. Flat bundle Laplacian is amenable since it can be factored into a synchronization step, a standard Laplacian, and a desynchronization step.

Consequently, the bundle Laplacian factors:

$$\mathcal{L}_{\mathcal{B}} = \mathbf{O}^T (\mathcal{L} \otimes \mathbf{I}_d) \mathbf{O},$$

where $\mathbf{O} \in \mathbb{R}^{nd \times nd}$ is block diagonal with v-th block being $\mathbf{O}_v$. We illustrate the differences between standard Laplacian, general bundle Laplacian, and flat bundle Laplacian in Figure 2. This factorization avoids the $\mathcal{O}\left(d^3|\mathsf{E}|\right)$ cost of computing the restriction map over each edge. Additionally, it allows to cast the bundle heat equation into a standard graph heat equation sandwiched between synchronization and desynchronization steps. This reduces the computation of the bundle heat kernel to that of the cheaper graph heat kernel, an operator that does not change depending on the bundle structure and can, therefore, also be pre-computed.

The model. The BuNN layer occurs in four steps, as illustrated in Figure 1 in which steps 1, 2, 3, and 4 also correspond to the Equations below. First, the bundle maps $\mathbf{O}_v$ are computed using a neural network ϕ and the use of Householder reflections. Second, an encoder step updates the initial node signals via a matrix $\mathbf{W} \in \mathbb{R}^{d \times d}$, and bias $\mathbf{b} \in \mathbb{R}^d$. Next, the features are diffused over the *learned* vector bundle. Finally, a non-linearity σ is applied. We summarize the steps in the following equations:

$$\mathbf{O}_v^{(\ell)} := \boldsymbol{\phi}^{(\ell)}(\mathsf{G}, \mathbf{X}^{(\ell)}, v) \quad \forall v \in \mathsf{V} \tag{1}$$

$$\mathbf{h}_v^{(\ell)} := \mathbf{O}_v^{(\ell)^T} \mathbf{W}^{(\ell)} \mathbf{O}_v^{(\ell)} \mathbf{x}_v^{(\ell)} + \mathbf{b}^{(\ell)} \quad \forall v \in \mathsf{V} \tag{2}$$

$$\mathbf{Z}^{(\ell+1)} := \mathcal{H}_{\mathcal{B}}(t)\mathbf{H}^{(\ell)} \tag{3}$$

$$\mathbf{X}^{(\ell+1)} := \sigma\left(\mathbf{Z}^{(\ell+1)}\right) \tag{4}$$

The diffusion time t in Equation 3 is a hyperparameter determining the scale at which messages are diffused. We provide additional algorithmic details in the Appendix (Section G).

Due to space consideration, we leave the theoretical analysis of BuNN to the Appendix (Sections C and D). There we provide formal statements and proofs supporting our claims on over-smoothing, over-squashing, and expressivity.

Comparison with Sheaf Neural Networks. Flat vector bundles are a special case of *cellular sheaves* (Curry, 2014; Bodnar et al., 2022), meaning that our model has close connections to Sheaf Neural Networks (SNNs) (Hansen & Gebhart, 2020; Bodnar et al., 2022; Barbero et al., 2022b;a; Battiloro et al., 2024). While most SNNs operate on fixed sheaves (Hansen & Gebhart, 2020; Barbero et al., 2022a; Battiloro et al., 2024), we focus on learning sheaves as in Neural Sheaf Diffusion (NSD) from Bodnar et al. (2022). BuNNs distinguish themselves from NSD in several ways. First, NSD approximates the heat equation using a time-discretized solution to the heat equation, which results in a standard message passing algorithm. In contrast, the direct use of the heat kernel allows BuNNs to *break away from the explicit message-passing paradigm*. Secondly, the use of flat bundles increases scalability since the bundle maps are computed at the node level. Additionally, flat bundles guarantees *path independence*, a requirement for the theory in Bodnar et al. (2022) to hold, often not satisfied for general sheaf constructions such as ones used in NSD. Thirdly, we allow ϕ to be any GNN while NSD restricts ϕ to be an MLP. We found that incorporating the graph structure in ϕ improved the experimental results. The update in Equation 2 is also different to NSD, and is necessary to prove our main theoretical result, Theorem D.3. We provide experimental comparisons to NSDs in the experimental section and show that BuNNs significantly outperform their sheaf counterparts. We provide a summarized comparison between GCNs, SNNs, and BuNNs in Table 4 of the Appendix.

4. Experiments

In this Section, we evaluate BuNNs on synthetic and real-world benchmarks. We provide supplementary information on the implementation and precise experimental details in the Appendix (Sections G and H respectively).

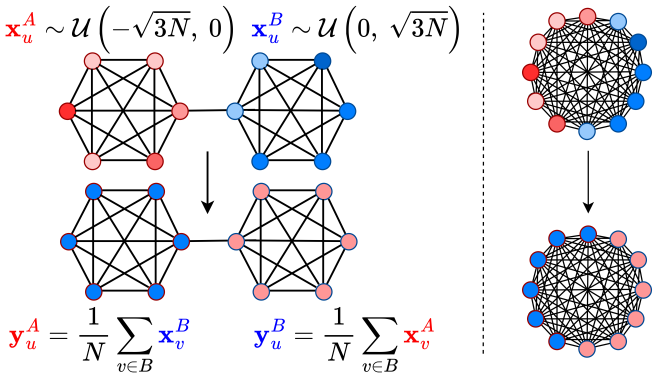

Figure 3. Synthetic over-squashing (left) and over-smoothing (right) tasks on barbell and clique graphs, respectively. In both cases, the blue nodes output the average over the red nodes and vice-versa.

4.1. Synthetic experiments: over-squashing and over-smoothing

Tasks. In this experiment, we propose two new synthetic tasks to test the capacity of models to alleviate over-squashing and over-smoothing. We compare BuNNs to MPNNs and simple baselines on these two new datasets. The tasks are node-regression tasks, in which nodes must average the input features of a subset of nodes as illustrated in Figure 3.

As a first task we evaluate the capacity of models to alleviate **over-squashing**. The graph is fixed to the barbell graph, consisting of two fully connected graphs and bridged by a single edge. Exchanging information between the two clusters is very challenging due to the bottleneck. The input features are sampled randomly in each cluster, with positive values in one cluster and negative in the other. The goal of the task is for each node to return the mean input over nodes from the opposite cluster. This requires information to be transferred across the bottleneck, making it a difficult task for models suffering from over-squashing.

As a second experiment, we test the capacity to mitigate **over-smoothing**. The graph is fully connected, in which all nodes are connected. The fully connected graph is a worst-case scenario for over-smoothing since after one step of message passing, the features are already fully averaged over the graph. This makes it challenging to predict different outputs over the different nodes. The input features are sampled randomly, half with positive values and half with negative values. The task is for positive nodes to return the mean output over the negative nodes and vice-versa.

Setup. As a first baseline, we consider a constant predictor always predicting 0, the expected mean over the whole graph, making it unaware of any cluster-specific information. As a second baseline, we consider the cluster-specific constant predictor predicting the expected mean over the opposite cluster, that is, $\frac{\pm\sqrt{3}}{2}$, depending on the cluster. Additionally, we consider GNN baselines to be a node-level MLP, GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), and NSD (Bodnar et al., 2022). The width of MPNNs is fixed to the minimal depth to avoid under-reaching, namely 3 for the Barbell and 1 for the fully connected graph, and ensure the width is large (> 128) considering the task. We compare these to a BuNN with an MLP learning the bundle maps of a comparable number of parameters and the same depth. We use Adam optimizer with 10^{-3} learning rate, batch size 1, and train for 500 epochs. We use 100 samples for training and 100 samples for testing.

Results. The results for $N = 10$ are reported in Table 1. All MPNNs perform poorly on the over-squashing task. This is explained by the fact that this task requires the exchange of information between nodes at a high resistance distance. In contrast, BuNNs solve this task perfectly since a larger hyper-parameter $t > 10$ allows to operate at a more appropriate scale of the graph. For the over-smoothing task, all MPNNs except NSD perform poorly. GCN and GAT perform particularly poorly, which is natural since they are the models with a formal proof of over-smoothing (Cai & Wang, 2020; Wu et al., 2023). As expected, both NSD and BuNNs solve the task perfectly, since they are designed to mitigate over-smoothing.

Table 1. Mean squared error (MSE) of different models for the over-squashing and over-smoothing experiments on the barbell and clique graphs, respectively.

Task	Barbell *(over-squashing)*	Clique *(over-smoothing)*
Base. 1	30.97 ± 0.42	30.94 ± 0.42
Base. 2	1.00 ± 0.07	0.99 ± 0.08
MLP	1.08 ± 0.07	1.10 ± 0.08
GCN	1.05 ± 0.08	29.65 ± 0.34
SAGE	0.90 ± 0.29	0.86 ± 0.10
GAT	1.07 ± 0.09	20.97 ± 0.40
NSD	1.09 ± 0.15	0.08 ± 0.02
BuNN	$\mathbf{0.01 \pm 0.07}$	$\mathbf{0.03 \pm 0.01}$

4.2. Real-world tasks

We evaluate BuNNs on the Long Range Graph Benchmark (Dwivedi et al., 2022) and the heterophilic tasks from Platonov et al. (2023). In our experiments, we report results both for the BuNN model described in Section 3 and a slight variation inspired by MixHop GNNs (Abu-El-Haija et al., 2019), which we call BuNN-Hop. We found this variation to perform exceptionally well on some real-world tasks.

Table 2. Results for the heterophilic tasks. Accuracy is reported for `roman-empire` and `amazon-ratings`, and ROC AUC is reported for `minesweeper`, `tolokers`, and `questions`. Best results are denoted by First and Second. Asterisk* denotes that some runs ran out of memory on an NVIDIA A10 GPU (24 GB).

	roman-empire	amazon-ratings	minesweeper	tolokers	questions
GCN	73.69 ± 0.74	48.70 ± 0.63	89.75 ± 0.52	83.64 ± 0.67	76.09 ± 1.27
SAGE	85.74 ± 0.67	53.63 ± 0.39	93.51 ± 0.57	82.43 ± 0.44	76.44 ± 0.62
GAT	80.87 ± 0.30	49.09 ± 0.63	92.01 ± 0.68	83.70 ± 0.47	77.43 ± 1.20
GAT-sep	88.75 ± 0.41	52.70 ± 0.62	93.91 ± 0.35	83.78 ± 0.43	76.79 ± 0.71
GT	86.51 ± 0.73	51.17 ± 0.66	91.85 ± 0.76	83.23 ± 0.64	77.95 ± 0.68
GT-sep	87.32 ± 0.39	52.18 ± 0.80	92.29 ± 0.47	82.52 ± 0.92	78.05 ± 0.93
NSD	80.41 ± 0.72	42.76 ± 0.54	92.15 ± 0.84	$78.83 \pm 0.76^*$	$69.69 \pm 1.46^*$
BuNN	91.75 ± 0.39	53.74 ± 0.51	98.99 ± 0.16	84.78 ± 0.80	78.75 ± 1.09
BuNN-Hop	89.63 ± 0.59	52.01 ± 0.45	97.91 ± 0.30	84.33 ± 0.77	78.68 ± 1.04

Heterophilic datasets. As we show in Sections 4.1 and C, BuNNs are provably capable of avoiding over-smoothing. It is, therefore, natural to test how BuNN performs on heterophilic graphs where over-smoothing is recognized as an important limitation (e.g., Yan et al. (2022)). We follow their methodology to evaluate BuNN on the 5 heterophilic tasks proposed in Platonov et al. (2023). We run the models with 10 different seeds and report the mean and standard deviation of the test accuracy for `roman-empire` and `amazon-ratings`, and mean and standard deviation test ROC AUC for `minesweeper`, `tolokers`, and `questions`. We use the standard baselines from Platonov et al. (2023), and also NSD from (Bodnar et al., 2022). We provide the hyper-parameters for our models in the Appendix (Section H.2).

We report the results in Table 2. BuNN achieves the best score on all tasks, with an average relative improvement of 4.4%. Its score on `minesweeper` is particularly impressive, which is significantly ahead of the rest and for which BuNN solves the task perfectly. We found that the optimal value of t over our grid search varies across datasets, being 1 for `amazon-ratings` and 100 for `roman-empire`. BuNN-Hop also performs similarly but tends slightly inferior to BuNN on 3 of the datasets and inferior on 2. BuNN consistently outperforms the sheaf-based model NSD. Such results showcase the strong modeling capacity of BuNN in heterophilic settings.

Long Range Graph Benchmark. In Section C.2, we show that BuNNs have desirable properties when it comes to over-squashing and modeling long-range interactions. To verify such claims empirically, we evaluate BuNN on tasks from the Long Range Graph Benchmark (LRGB) (Dwivedi et al., 2022). We consider the `Peptides` dataset consisting of 15 535 graphs which come with two associated graph-level tasks, `Peptides-func` and `Peptides-struct`. The graph classification task in `Peptides-func` is to predict the function of the peptide from 10 classes, while the regression task in `Peptides-struct` is inferring the 3D properties of the peptides. In both cases, we follow the standard experimental setup detailed by Dwivedi et al. (2022) alongside the updated suggestions from Tönshoff et al. (2023). Baseline models are taken from Tönshoff et al. (2023) and include MPNNs, transformers, and the current SOTA models (Gutteridge et al., 2023; He et al., 2023).

Table 3. Results for the `Peptides-struct` and `Peptides-func` tasks from the Long Range Graph Benchmark (results are $\times 100$ for clarity). The best result is **bold**.

`Peptides`	`-func`	`-struct`
Model	**Test AP** $\uparrow$	**Test MAE** $\downarrow$
GCN	68.60 ± 0.50	24.60 ± 0.07
GINE	66.21 ± 0.67	24.73 ± 0.17
GatedGCN	67.65 ± 0.47	24.77 ± 0.09
GPS	65.34 ± 0.91	25.09 ± 0.14
DReW	71.50 ± 0.44	25.36 ± 0.15
GAPH ViT	69.42 ± 0.75	$\mathbf{24.49} \pm 0.16$
BuNN	71.03 ± 0.22	24.98 ± 0.12
BuNN-Hop	$\mathbf{71.92} \pm 0.22$	24.63 ± 0.25

We report the results in Table 3. BuNNs achieve, to the best of our knowledge, a new state-of-the-art result on `Peptides-func`. The results on `Peptides-struct` are also competitive, with BuNN-Hop remaining within a standard deviation from the best result. The strong performance provides further evidence of the long-range capabilities of BuNNs.

5. Conclusion

By proposing Bundle Neural Networks, we show that geometric structures are helpful to overcome the current limitations of existing graph machine learning methods. We showed, through synthetic experiments, that message diffusion on bundles can mitigate issues such as over-smoothing, and over-squashing. Finally, we showed that BuNNs perform well on a range real-world benchmarks.

References

torch-householder: Efficient Householder transformation in PyTorch. URL https://www.obukhov.ai/torch-householder.html.

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 21–29. PMLR, May 2019.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

Azizian, W. and marc lelarge. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021.

Barbero, F., Bodnar, C., Borde, H. S. d. O., Bronstein, M., Veličković, P., and Liò, P. Sheaf Neural Networks with Connection Laplacians. In *Proceedings of Topological, Algebraic, and Geometric Learning Workshops 2022*, pp. 28–36. PMLR, November 2022a.

Barbero, F., Bodnar, C., Borde, H. S. d. O., and Lio', P. Sheaf Attention Networks. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*, 2022b.

Battiloro, C., Wang, Z., Riess, H., Di Lorenzo, P., and Ribeiro, A. Tangent bundle convolutional learning: From manifolds to cellular sheaves and back. *IEEE Transactions on Signal Processing*, 72:1892–1909, 2024. doi: 10.1109/TSP.2024.3379862.

Bodnar, C., Giovanni, F. D., Chamberlain, B. P., Lio, P., and Bronstein, M. M. Neural Sheaf Diffusion: A Topological Perspective on Heterophily and Oversmoothing in GNNs. In *Advances in Neural Information Processing Systems*, 2022.

Cai, C. and Wang, Y. A Note on Over-Smoothing for Graph Neural Networks. In *ICML Graph Representation Learning workshop*. arXiv, 2020.

Curry, J. M. *Sheaves, cosheaves and applications*. University of Pennsylvania, 2014.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

Di Giovanni, F., Rowbottom, J., Chamberlain, B. P., Markovich, T., and Bronstein, M. M. Graph Neural Networks as Gradient Flows: understanding graph convolutions via energy, October 2022.

Di Giovanni, F., Giusti, L., Barbero, F., Luise, G., Lio', P., and Bronstein, M. On Over-Squashing in Message Passing Neural Networks: The Impact of Width, Depth, and Topology, February 2023. arXiv:2302.02941 [cs, stat].

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Geerts, F. and Reutter, J. L. Expressiveness and approximation properties of graph neural networks. In *International Conference on Learning Representations*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1263–1272. PMLR, July 2017.

Gutteridge, B., Dong, X., Bronstein, M., and Di Giovanni, F. Drew: dynamically rewired message passing with delay. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

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, Red Hook, NY, USA, 2017. Curran Associates Inc.

Hansen, J. and Gebhart, T. Sheaf Neural Networks. In *NeurIPS Workshop TDA and Beyond*, 2020.

He, X., Hooi, B., Laurent, T., Perold, A., LeCun, Y., and Bresson, X. A generalization of vit/mlp-mixer to graphs. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

Li, Q., Han, Z., and Wu, X.-M. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 3538–3545. Association for the Advancement of Artificial Intelligence, February 2018. ISBN 978-1-57735-800-8.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33 (01):4602–4609, July 2019.

Obukhov, A. Efficient Householder transformation in PyTorch, 2021. URL https://github.com/toshas/torch-householder.

Oono, K. and Suzuki, T. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*, 2020.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*, 2023.

Rosenbluth, E., Tönshoff, J., and Grohe, M. Some Might Say All You Need Is Sum. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 4172–4179, Macau, SAR China, August 2023. International Joint Conferences on Artificial Intelligence Organization.

Rosenbluth, E., Tönshoff, J., Ritzert, M., Kisin, B., and Grohe, M. Distinguished in uniform: Self-attention vs. virtual nodes. In *The Twelfth International Conference on Learning Representations*, 2024.

Scarselli, F., Gori, M., Tsoi, A., Hagenbuchner, M., and Monfardini, G. The Graph Neural Network Model. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 20:61–80, January 2009.

Singer, A. and Wu, H.-T. Vector diffusion maps and the connection Laplacian. *Communications on Pure and Applied Mathematics*, 65(8):1067–1144, 2012.

Sperduti, A. Encoding Labeled Graphs by Labeling RAAM. In *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993.

Tönshoff, J., Ritzert, M., Rosenbluth, E., and Grohe, M. Where did the gap go? reassessing the long-range graph benchmark. In *The Second Learning on Graphs Conference*, 2023.

Topping, J., Giovanni, F. D., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Wang, X. and Zhang, M. How Powerful are Spectral Graph Neural Networks, June 2022.

Wu, X., Ajorlou, A., Wu, Z., and Jadbabaie, A. Demystifying Oversmoothing in Attention-Based Graph Neural Networks. *Advances in Neural Information Processing Systems*, 36:35084–35106, December 2023.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2019.

Yan, Y., Hashemi, M., Swersky, K., Yang, Y., and Koutra, D. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 1287–1292, Los Alamitos, CA, USA, dec 2022. IEEE Computer Society.

A. Additional Background

GNNs and feature transformations. A $\mathrm{GNN}_{\mathbf{\Theta}}$ is a (continuous) map parameterized by $\mathbf{\Theta}$ that takes as input a graph alongside node signals $\mathsf{G} = (\mathsf{V}, \mathsf{E}, \mathbf{X})$ and outputs a transformed signal $(\mathsf{V}, \mathsf{E}, \mathbf{X}')$. A GNN on a graph G is therefore a feature transformation $\mathrm{GNN}_{\Theta} : \mathbb{R}^{n\times c} \to \mathbb{R}^{n\times c'}$. Given a collection of graphs $\mathcal{G}$, with feature space $\mathbb{R}$, a feature transformation F of $\mathcal{G}$ is an assignment of every graph $\mathsf{G} \in \mathcal{G}$ to a feature transformation $F_G : \mathbb{R}^{n_G\times c} \to \mathbb{R}^{n_G\times c'}$. The set of (continuous) feature transformations over a collection of graphs in $\mathcal{G}$ is denoted $\mathcal{C}\left(\mathcal{G}, \mathbb{R}^c, \mathbb{R}^{c'}\right)$.

Cellular sheaves. A *cellular sheaf* (Curry, 2014) $(\mathcal{F}, \mathsf{G})$ over an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ augments G by attaching to each node v and edge e a vector space space called *stalks* and denoted by $\mathcal{F}(v)$ and $\mathcal{F}(e)$, usually the stalks are copies of $\mathbb{R}^d$ for some d. Additionally, every incident node-edge pair $v \trianglelefteq e$ gets assigned a linear map between stalks called *restriction maps* and denoted $\mathcal{F}_{v\trianglelefteq e} : \mathcal{F}(v) \to \mathcal{F}(e)$. Given two nodes v and u connected by an edge $(v,\ u)$, we can *transport* a vector $\mathbf{x}_v \in \mathcal{F}(v)$ from v to u by first mapping it to the stalk at $e = (v, u)$ using $\mathcal{F}_{v\trianglelefteq e}$, and mapping it to $\mathcal{F}(u)$ using the transpose $\mathcal{F}^T_{u\trianglelefteq e}$. As a generalization of the graph adjacency matrix, the *sheaf adjacency matrix* $\mathbf{A}_{\mathcal{F}} \in \mathbb{R}^{nd\times nd}$ is defined as a block matrix in which each $d \times d$ block $(\mathbf{A}_{\mathcal{F}})_{uv}$ is $\mathcal{F}^T_{u\trianglelefteq e}\mathcal{F}_{v\trianglelefteq e}$ if there is an edge between u and v and $\mathbf{0}_{d\times d}$ if there is not. We similarly define the block diagonal *degree matrix* $\mathbf{D}_{\mathcal{F}} \in \mathbb{R}^{nd\times nd}$ as $(\mathbf{D}_{\mathcal{F}})_{vv} := d_v\mathbf{I}_{d\times d}$. Finally, we define the sheaf Laplacian as $\mathbf{L}_{\mathcal{F}} = \mathbf{D}_{\mathcal{F}} - \mathbf{A}_{\mathcal{F}}$. These matrices act as bundle generalizations of their well-known standard graph counterparts and we recover such matrices when $\mathcal{F}(v) \cong \mathbb{R}$ and $\mathcal{F}_{v\trianglelefteq e}v = 1$ for all $v \in \mathsf{V}$ and $e \in \mathsf{E}$.

B. Comparison to other models

In this Section we compare BuNNs to Graph Convolution Networks and Sheaf Neural Networks.

Link to Graph Convolutional Networks. It is possible to derive Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) as an approximation of BuNNs operating over a trivial bundle. Setting $t = 1$ Equation 3 becomes $\mathbf{Z}^{(l+1)} = \exp(-\mathcal{L}_{\mathsf{G}})\mathbf{H}^{(l)}$. The approximation $\exp(-\mathcal{L}_{\mathsf{G}}) \approx 1 - \mathcal{L}_{\mathsf{G}}$ gives the update $\mathbf{Z}^{(l+1)} = (1 - \mathcal{L}_{\mathsf{G}})\mathbf{H}^{(l)} = \mathbf{A}_{\mathsf{G}}\mathbf{H}^{(l)}$ recovering the GCN update. Section C.1 shows that BuNNs can avoid over-smoothing, highlighting an important difference between GCNs and BuNNs.

Table 4. Comparison between GCN, SNN, and BuNN in terms of propagation and message type, as well as their capability of mitigating issues related to GNNs.

	GCN	**SNN**	**BuNN**
Propagation	$\mathbf{A}_{\mathsf{G}}\mathbf{X}$	$\mathcal{L}_{\mathcal{F}}\mathbf{X}$	$\mathcal{H}_{\mathcal{B}}(t)\mathbf{X}$
Message type	Standard (Local)	Standard (Local)	Diffusive (Global)
No under-reaching	✗	✗	✓
No over-squashing	✗	✗	✓
No over-smoothing	✗	✓	✓

C. Properties of Bundle Neural Networks

In this Section, we give a formal analysis of the BuNN model. In Section C.1, we derive the fixed points of the bundle heat diffusion, which is the subspace of signals towards which solutions converges exponentially fast, and show that despite converging to fixed points, BuNNs can retain information at the node level and therefore avoid over-smoothing. Section C.2 discusses how our model can capture long-range interactions and mitigate over-squashing.

First, we show that the heat equation over flat vector bundles can be casted in terms of a standard graph heat equation:

Lemma C.1. *For every node v, the solution at time t of the heat equation on a connected bundle $\mathsf{G} = (\mathsf{V},\ \mathsf{E},\ \mathbf{O})$ with input node features $\mathbf{X}$ satisfies:*

$$(\mathcal{H}_{\mathcal{B}}(t)\mathbf{X})_v = \sum_{u\in\mathsf{V}} \mathcal{H}(t, v, u)\mathbf{O}_v^T\mathbf{O}_u\mathbf{x}_u,$$

where $\mathcal{H}(t)$ is the standard graph heat kernel, and $\mathcal{H}(t,\ v,\ u) \in \mathbb{R}$ its the entry at $(v,\ u)$.

C.1. Fixed points and over-smoothing.

A significant limitation of MPNNs is the problem of *over-smoothing*, the phenomenon of node features becoming increasingly indistinguishable with the increase of the MPNN's depth. Over-smoothing has been repeatedly highlighted as one of the critical problems in GNNs, making training deep GNNs much more challenging. Such behavior is related to the diffusion used in MPNNs being similar to that of heat diffusion on graphs (Di Giovanni et al., 2022), which converges to uninformative fixed points[1] resulting in a loss of information at the node level. Bodnar et al. (2022) show that the richer sheaf (or bundle) structure, however, gives rise to more interesting limiting behavior. Indeed, by Lemma C.1, since $\lim_{t\to\infty} \mathcal{H}(t,\ v,\ u) = \frac{d_u}{2|\mathsf{E}|}$, the limit over time of a solution is $\frac{1}{2|E|}\sum_{u\in\mathsf{V}} d_u \mathbf{O}_v^T \mathbf{O}_u \mathbf{x}_u$. To better understand the space of fixed points, notice that any signal $\mathbf{Y} \in \mathbb{R}^{n\times d}$ expressible as $\mathbf{y}_v = \frac{1}{2|E|}\sum_{u\in\mathsf{V}} d_u \mathbf{O}_v^T \mathbf{O}_u \mathbf{x}_u$ for some $\mathbf{X}$ is a fixed point. For any two nodes $u,\ v$ we get:

$$\mathbf{O}_v \mathbf{y}_v = \frac{1}{2|E|}\sum_{w\in\mathsf{V}} d_w \mathbf{O}_w \mathbf{x}_w = \mathbf{O}_u \mathbf{y}_u. \tag{5}$$

Consequently, the fixed points of vector diffusion have a global geometric dependency where all nodes relate to each other by some orthogonal transformation, e.g. the output in Figure 1.

Equation 5 provides insight into the smoothing properties of BuNNs. When the bundle is trivial, the equation reduces to $\mathbf{y}_v = \mathbf{y}_u$, with no information at the node level, i.e. over-smoothing. When it is non-trivial, the output signal can vary across the graph (e.g., $\mathbf{y}_v \neq \mathbf{y}_u$ for two nodes u and v). We summarize this in Proposition C.2.

Proposition C.2. *Let $\mathbf{Y}$ be the output of a BuNN layer with $t = \infty$, where G is a connected graph, and the bundle maps are not all equal. Then, there exists $u, v \in \mathsf{V}$ connected such that $\mathbf{y}_v \neq \mathbf{y}_u$.*

C.2. Over-squashing and long range interactions.

While message-passing in MPNNs constitutes a strong inductive bias, this local mechanism is problematic when the task requires the MPNN to capture interactions between distant nodes. These issues have been attributed mainly to the *over-squashing* of information (Alon & Yahav, 2021). Topping et al. (2022) and Di Giovanni et al. (2023) formalize over-squashing through a sensitivity analysis, giving upper bounds on the partial derivatives of the outputs of the MPNN with respect to the input. In particular, they show that under weak assumptions on the message function, the Jacobian of MPNNs satisfies

$$\left|\partial \left(\mathrm{MPNN}_\Theta(\mathbf{X})\right)_u / \partial \mathbf{x}_v\right| \leq c^r \left(\mathbf{A}^r\right)_{uv}, \tag{6}$$

for any nodes $u,\ v$, where r is the depth of the network and c is a constant. Given two nodes u, v at a distance r, message-passing will require at least r layers for the two nodes to communicate and overcome *under-reaching*. If r is large, distant nodes can communicate, but Di Giovanni et al. (2023) show that over-squashing becomes dominated by vanishing gradients. Built on top of such sensitivity analysis, we compute the Jacobian for BuNN:

Lemma C.3. *Let BuNN be a linear layer defined by Equations 2 & 3 with hyperparameter t. Then, for any connected graph and nodes $u,\ v$, we have*

$$\frac{\partial \left(\mathrm{BuNN}\left(\mathbf{X}\right)\right)_u}{\partial \mathbf{x}_v} = \mathcal{H}(t,u,v)\mathbf{O}_u^T \mathbf{W} \mathbf{O}_v,$$

The form of the Jacobian in Lemma C.3 differs significantly from the usual form in Equation 6. First, all nodes communicate in a single BuNN layer since $\mathcal{H}(t,u,v) > 0$ for all u, v, and t, allowing for *direct pair-wise communication between nodes*, making a BuNN layer operate globally similarly to the attention mechanism in the Transformer model, and therefore overcome under-reaching. Secondly, taking t to be large allows BuNNs to operate on a larger scale, allowing stronger communication between distant nodes and overcoming over-squashing without the vanishing gradient problem. Indeed, $\mathcal{H}(t,u,v)$ converges to $\frac{d_u}{2|\mathsf{E}|}$ exponentially fast with $t \to \infty$, showing that for larger scales the sensitivity between two nodes does not depend on how they are connected.

Further, Lemma C.3 gives a finer picture of the capabilities of BuNNs. For example, to mitigate over-squashing a node may decide to ignore information received from certain nodes while keeping information received from others. This allows the model to reduce the receptive field of certain nodes. We formalize such a behavior in Corollary C.4.

[1] Up to degree scaling if using the symmetric-normalized Laplacian, see Li et al. (2018).

Corollary C.4. *Consider n nodes $u,\ v,$ and w_i, for $i = 1, \dots n-2$, of a connected graph with 2 dimensional bundle such that $\mathbf{x}_v \neq \mathbf{x}_{w_i}$ and $\mathbf{x}_u \neq \mathbf{x}_{w_i}\ \forall i$. Then in a BuNN layer, at a given channel, the node v can learn to ignore the information from all w_is while keeping information from u.*

D. Expressivity of the model

In this Section, we study the expressive power of BuNNs at the *node level* from a function approximation perspective. We note that such expressive power is strictly stronger than the more common graph-level expressivity, which usually relates GNNs to the Weisfeiler-Leman hierarchy. In node-level approximation, we desire our model to be able to approximate *arbitrary feature transformations over the nodes*. Further, existing work generally focuses on a fixed graph or graphs of bounded size (Azizian & marc lelarge, 2021; Geerts & Reutter, 2022; Wang & Zhang, 2022). In contrast, we focus on the more challenging case of potentially infinite families of graphs, known as *uniform approximation* (Rosenbluth et al., 2023). Such analysis studies the class of feature transformation that GNN architectures can approximate over a potentially *infinite collection* of graphs $\mathcal{G}$. Given input and output dimensions c and c', a family of GNNs will parameterize a subset of continuous feature transformation $\mathcal{C}(\mathcal{G}, \mathbb{R}^c, \mathbb{R}^{c'})$ over $\mathcal{G}$. It is natural to ask which feature transformations a specific family of GNNs can or cannot parameterize. To this end, we introduce the notion of *compact uniform approximation* as an amendment to the definition of uniform approximation from Rosenbluth et al. (2023) and discuss our specific choice in the Appendix (Section F).

Definition D.1. Let $\mathcal{F} \subseteq \mathcal{C}(\mathcal{G}, \mathbb{R}^c, \mathbb{R}^{c'})$ be a set of feature transformations over a family of graphs $\mathcal{G}$, and let $H \in \mathcal{C}(\mathcal{G}, \mathbb{R}^c, \mathbb{R}^{c'})$ a feature transformation over $\mathcal{G}$. We say that $\mathcal{F}$ *compactly uniformly approximates* H, if for all finite subsets $\mathcal{K} \subseteq \mathcal{G}$, for all compact $K \subset \mathbb{R}^c$, and for all $\epsilon > 0$, there exists an $F \in \mathcal{F}$ such that for all $\mathsf{G} \in \mathcal{K}$ and $\mathbf{X} \in K^{n_\mathsf{G}}$, we have that $||F_\mathsf{G}(\mathbf{X}) - H_\mathsf{G}(\mathbf{X})||_\infty \leq \epsilon$.

As we show in Proposition D.2, bounded-depth MPNNs fail to have this property for any arbitrary $\mathcal{G}$, since fixing the depth of the MPNNs to ℓ, we can take a single compactly featured graph $\mathsf{G} \in \mathcal{G}$ with diameter larger than ℓ. As there is a node whose receptive field does not include all nodes in such a G, the architecture cannot uniformly approximate every function on $\mathcal{G}$.[2]

Proposition D.2. *There exists a family $\mathcal{G}$ consisting of connected graphs such that bounded-depth MPNNs are not compact uniform approximators, even if enriched with unique positional encoding.*

It was shown in Rosenbluth et al. (2024) that *even with injective Positional Encodings (PEs)*, there are functions that MPNNs with virtual-nodes cannot approximate uniformly while transformers can, and vice-versa. Instead, we prove that our model is compactly uniformly universal given injective PEs. To the best of our knowledge, this is the first positive uniform approximation result for a GNN, demonstrating the remarkable modeling capabilities of BuNNs.

Theorem D.3. *Let $\mathcal{G}$ be a set of connected graphs with injective positional encodings. Then 2-layer deep BuNNs with encoder/decoder have compact uniform approximation over $\mathcal{G}$.*

In particular, let $\epsilon > 0$ and H be a feature transformation on a finite subset $\mathcal{K} \subseteq \mathcal{G}$ and $K \subseteq \mathbb{R}^d$ a compact set, then there is a 2-layer deep BuNN with width of order $\mathcal{O}\left(\sum_{\mathsf{G}\in\mathcal{K}} |\mathsf{V}_\mathsf{G}|\right)$ that approximates H over $\bigsqcup_{\mathsf{G}\in\mathcal{K}} K^{n_\mathsf{G}} \subseteq \bigsqcup_{\mathsf{G}\in\mathcal{G}} \mathbb{R}^{n_\mathsf{G} d}$. In other words, the required hidden dimension of BuNN is only linearly dependent on the number of nodes in the family of graphs.

E. Proofs

In this Section, we provide proof of the theoretical results from the main text. Namely Lemma C.1, Proposition C.2, Lemma C.3, Corollary C.4, Proposition D.2, and finally Theorem D.3.

Lemma C.1. For every node v, the solution at time t of heat diffusion on a connected bundle $\mathsf{G} = (\mathsf{V},\ \mathsf{E},\ \mathbf{O})$ with input node features $\mathbf{X}$ satisfies:

$$\left(\mathcal{H}_\mathcal{B}(t)\mathbf{X}\right)_v = \sum_{u\in\mathsf{V}} \mathcal{H}(t, v, u)\mathbf{O}_v^T\mathbf{O}_u\mathbf{x}_u, \tag{7}$$

where $\mathcal{H}(t)$ is the standard graph heat kernel, and $\mathcal{H}(t,\ v,\ u) \in \mathbb{R}$ its the entry at $(v,\ u)$.

Proof. Since $\mathcal{L}_\mathcal{B} = \mathbf{O}_\mathcal{B}^T\mathcal{L}\mathbf{O}_\mathcal{B}$ we get $\mathcal{H}_\mathcal{B}(t, u, v) = \mathbf{O}_\mathcal{B}^T\mathcal{H}(t, u, v)\mathbf{O}_\mathcal{B}$. □

[2] This phenomenon in MPNNs is often called 'under-reaching'.

E.1. Over-smoothing: proofs of Section C.1.

In this section, we prove the result of over-smoothing. This result follows straightforwardly from Lemma C.1.

Proposition C.2. Let $\mathbf{Y}$ be the output of a BuNN layer with $t = \infty$, where G is a connected graph, and the bundle maps are not all equal. Then, $u, v \in \mathsf{V}$ is connected such that $\mathbf{y}_v \neq \mathbf{y}_u$ almost always.

Proof. Consider a stable signal $\mathbf{X} \in \ker \mathcal{L}_{\mathcal{B}}$ and pick $u, v \in \mathsf{V}$ such that $\mathbf{O}_u \neq \mathbf{O}_v$. As $\mathbf{X}$ is stable, it must have 0 Dirichlet energy, so we must have that $\mathbf{O}_u\mathbf{y}_u = \mathbf{O}_v\mathbf{y}_v$, but as $\mathbf{O}_u \neq \mathbf{O}_v$ we have that $\mathbf{y}_u \neq \mathbf{y}_v$, which holds except in degenerate cases such as when the matrices $\mathbf{O}_u$ are reducible, or when the original signal is the zero vector. □

E.2. Over-squashing: proofs of Section C.2.

In this Section we prove our results on over-squashing and long-range interactions. The Jacobian result follows straightforwardly from Lemma C.1 and the definition of a BuNN layer, and the Corollary follows from the Jacobian result.

Lemma C.3. Let BuNN be a linear layer defined by Equations 2 & 3. Then, for any connected graph and nodes $u,\ v$, we have

$$\frac{\partial \left(\mathrm{BuNN}\left(\mathbf{X}\right)\right)_u}{\partial \mathbf{x}_v} = \mathcal{H}(t,u,v)\mathbf{O}_u^T\mathbf{W}\mathbf{O}_v,$$

and therefore

$$\lim_{t\to\infty} \frac{\partial \left(\mathrm{BuNN}\left(\mathbf{X}\right)\right)_u}{\partial \mathbf{x}_v} = \frac{d_v}{2|\mathsf{E}|}\mathbf{O}_u^T\mathbf{W}\mathbf{O}_v.$$

Proof. The result follows from the closed-form solution of the heat kernel from Lemma C.1. We start by applying the bundle encoder from Equation 2 that updates each node representation as $\mathbf{h}_v = \mathbf{O}_v^T\mathbf{W}\mathbf{O}_v\mathbf{x}_v + \mathbf{b}$. We therefore compute

$$\frac{\partial \left(\mathrm{BuNN}\left(\mathbf{X}\right)\right)_u}{\partial \mathbf{x}_v} = \frac{\partial}{\partial \mathbf{x}_v}\left[\sum_{v\in\mathsf{V}} \mathcal{H}(t,u,v)\mathbf{O}_u^T\mathbf{O}_v\left(\mathbf{O}_v^T\mathbf{W}\mathbf{O}_v\mathbf{x}_v + \mathbf{b}\right)\right] \tag{8}$$

$$= \mathcal{H}(t,u,v)\mathbf{O}_u^T\mathbf{W}\mathbf{O}_v. \tag{9}$$

The second statement follows from the fact that $\mathcal{H}(t,u,v) \to \frac{d_u}{2|E|}$ □

To illustrate the flexibility of such a result, we examine a setting in which we want nodes to select which nodes they receive information from, therefore 'reducing' their receptive field.

Corollary C.4. Consider n nodes $u,\ v,$ and $w_i,$ for $i = 1, \dots n-2$ of a connected graph G with 2 dimensional features such that $\mathbf{x}_v \neq \mathbf{x}_{w_i}$ and $\mathbf{x}_u \neq \mathbf{x}_{w_i}$ $\forall i$. Assume the bundle structure is learned with a node-level MLP. Then, in a given channel, node v can learn to ignore the information from all $w_i s$ while keeping information from u.

Proof. We denote $\mathbf{y}$ the output of the layer, and index the two dimensions by super-scripts, i.e. $\mathbf{y} = \begin{pmatrix}\mathbf{y}^{(1)}\\ \mathbf{y}^{(2)}\end{pmatrix}$. Our goal is to have $\frac{\partial \mathbf{y}_v^{(1)}}{\partial \mathbf{y}_u} \neq \begin{pmatrix}0 & 0\end{pmatrix}$, while $\frac{\partial \mathbf{y}_v^{(1)}}{\partial \mathbf{y}_{w_i}} = \begin{pmatrix}0 & 0\end{pmatrix}$ for all i. This would make the first channel of the output at v insensitive to the input at all w_is while being sensitive to the input at node u.

Fix $\mathbf{O}_v = \mathbf{O}_u = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$ and $\mathbf{O}_{w_i} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$. Such maps can always be learned by an MLP, by the assumptions on $\mathbf{x}_v$, $\mathbf{x}_u$, and $\mathbf{x}_{w_i}$ and by the universality of MLPs. Let the weight matrix be $\mathbf{W} = \begin{pmatrix}w_{11} & w_{12}\\ w_{21} & w_{22}\end{pmatrix}$. By Lemma C.3 we get $\frac{\partial \mathbf{y}_v}{\partial \mathbf{x}_u} = \mathcal{H}(t,v,u)\mathbf{O}_v^T\mathbf{W}\mathbf{O}_u = \mathcal{H}(t,v,u)\begin{pmatrix}w_{22} & w_{12}\\ w_{21} & w_{11}\end{pmatrix}$ and $\frac{\partial \mathbf{y}_v}{\partial \mathbf{x}_{w_i}} = \mathcal{H}(t,v,w_i)\mathbf{O}_v^T\mathbf{W}\mathbf{O}_{w_i} = \mathcal{H}(t,v,w_i)\begin{pmatrix}w_{21} & w_{22}\\ w_{11} & w_{12}\end{pmatrix}$. Setting w_{21} and w_{22} to 0 gives $\frac{\partial \mathbf{y}_v^{(1)}}{\partial \mathbf{x}_{w_i}} = \begin{pmatrix}0 & 0\end{pmatrix}$ and $\frac{\partial \mathbf{y}_v^{(1)}}{\partial \mathbf{x}_u} = \mathcal{H}(t,v,u)\begin{pmatrix}0 & w_{12}\end{pmatrix} \neq \mathbf{0}$, as desired. □

E.3. Expressivity of BuNNs: proofs of Section D.

We now turn to BuNN's expressivity. Before proving that BuNNs have compact uniform approximation, we prove that MPNNs fail to have this property. This proves BuNNs' superiority and shows that uniform expressivity is a good theoretical framework for comparing GNN architectures.

Proposition D.2. There exists a family $\mathcal{G}$ consisting of connected graphs such that bounded-depth MPNNs are not compact uniform approximators, even if enriched with unique positional encoding.

Proof. Let $\mathcal{G}$ be any family of connected graphs with an unbounded diameter (for example, the $n \times n$ grids with $n \to \infty$). Let the depth of the MPNN be L. Let $\mathsf{G} \in \mathcal{G}$ be a graph with diameter $> L$, and let u and v be two nodes in V_{G} at distance $> L$. Note that the output at v will be insensitive to the input at u, and therefore, the MPNN cannot capture feature transformations where the output at v depends on the input at u. This argument holds even when nodes are given unique positional encodings. □

We now turn to our main theoretical contribution. The proof of Theorem D.3 is split into two parts. The first proves that 1-layer BuNNs have compact uniform approximation over *linear feature transformations*. The second part is extending to continuous feature transformation, which is an application of classical results.

We start by recalling what a linear feature transformation over a family of graphs $\mathcal{G}$ is:

Definition E.1. A linear feature transformation $L \in \mathcal{C}(\mathcal{G}, \mathbb{R}^c, \mathbb{R}^{c'})$ over a family of graphs $\mathcal{G}$ is an assignment of each graph $\mathsf{G} \in \mathcal{G}$ to a linear map $L_{\mathsf{G}} : \mathbb{R}^{n_{\mathsf{G}} c} \to \mathbb{R}^{n_{\mathsf{G}} c'}$. Here, linearity means that for any two node-signals $\mathbf{X}_1 \in \mathbb{R}^{nc}$ and $\mathbf{X}_2 \in \mathbb{R}^{nc'}$, and any real number $\alpha \in \mathbb{R}$, it holds that $L_{\mathsf{G}}(\alpha \mathbf{X}_1) = \alpha L_{\mathsf{G}}(\mathbf{X}_1)$, and $L_{\mathsf{G}}(\mathbf{X}_1 + \mathbf{X}_2) = L_{\mathsf{G}}(\mathbf{X}_1) + L_{\mathsf{G}}(\mathbf{X}_2)$.

In the proof, we use the following Theorem, which adapts classical results about the universality of MLPs.

Theorem E.2. *If a class of neural networks has compact uniform approximation over $\mathcal{G}$ with respect to linear functions and contains non-polynomial activations, then it has compact universal approximation over $\mathcal{G}$ with respect to continuous functions.*

Proof. Classical theorems such as Theorem 1 in (Cybenko, 1989) allow us to approximate any continuous function over a compact set by composing a linear map C, an activation σ, and an affine map $A \cdot + b$. By assumption, we can implement the linear map, the activation, and the affine map; hence, by composing them, we can approximate any continuous function over the compact set. □

We are now ready to prove the paper's main result: that, given injective positional encodings, BuNNs are compact universal approximators.

Theorem D.3. Let $\mathcal{G}$ be a set of connected graphs with injective positional encodings. Then there is a 2-layer deep BuNN with compact uniform approximation over $\mathcal{G}$.

In particular, let $\epsilon > 0$ and h be a feature transformation supported on $\bigsqcup_{\mathsf{G} \in \mathcal{K}} K^{n_{\mathsf{G}}} \subseteq \bigsqcup_{\mathsf{G} \in \mathcal{G}} \mathbb{R}^{n_{\mathsf{G}} d}$ with $\mathcal{K} \subseteq \mathcal{G}$ finite and $K \subseteq \mathbb{R}^d$ a compact set, then there is a 2-layer deep BuNN with width $\mathcal{O}\left(\sum_{\mathsf{G} \in \mathcal{K}} |\mathsf{V}_{\mathsf{G}}|\right)$ that approximates h with uniform error $< \epsilon$.

Proof. **Reducing to linear approximation.** It suffices to show that a BuNN layer can approximate any linear feature transformation L because we can apply classical results such as Theorem E.2 to get universal approximation of 2-layer deep networks with activation. Following Definition D.1, we aim to show that we can approximate a linear feature transformation L on any compact subset. For this, we fix $\epsilon > 0$, the finite subset $\mathcal{K} \subseteq \mathcal{G}$, and compact feature space $K \subseteq \mathbb{R}^c$. In fact, we assume that $K = \mathbb{R}^c$ since approximating a linear map on any compact feature space is equivalent to approximating it on the whole space because a linear map defined on a neighborhood of the 0 vector can be extended uniquely to the whole vector space. Our goal is therefore to find a parameterization of a single BuNN layer such that for any graph $\mathsf{G} \in \mathcal{K}$ and for any input feature $\mathbf{X} \in \mathbb{R}^{n_{\mathsf{G}} c}$, we have $\|L_{\mathsf{G}}(X) - \mathrm{BuNN}_{\mathsf{G}}(\mathbf{X})\|_\infty < \epsilon$. We will show that L can be parameterized exactly. Since L is linear, it suffices to find a linear BuNN layer that satisfies for any $\mathsf{G} \in \mathcal{K}$ and any $\mathbf{X} \in \mathbb{R}^{n_{\mathsf{G}} c}$, $\frac{\partial (L(\mathbf{X}))_u}{\partial \mathbf{x}_v} = \frac{\partial (\mathrm{BuNN}\, \mathbf{X})_u}{\partial \mathbf{x}_v}$. By Lemma C.3, we have $\frac{\partial\, \mathrm{BuNN}(\mathbf{X})_u}{\partial \mathbf{x}_v} = \mathcal{H}(t, u, v) \mathbf{O}_u \mathbf{W} \mathbf{O}_v^T$. Hence, since MLPs are universal and the positional encodings are injective, it suffices to find bundle maps $\mathbf{O} : \mathsf{V} \to O(k)$ and $\mathbf{W}$ such that $\frac{1}{n_{\mathsf{G}} d_u} \sum_{v \in \mathsf{V}} \mathbf{O}_u^T \mathbf{W} \mathbf{O}_v = \frac{\partial (LX)_u}{\partial X_v}$ for every $u, v \in \mathsf{V}$.

Defining the encoder and decoder: In order to find such a BuNN, we first need a linear encoder $\text{lift} : \mathbb{R}^c \to \mathbb{R}^{2ck}$ which will be applied at every node before applying a $2ck$ dimensional BuNN layer. The lifting transformation maps each node vector $\mathbf{X}_u$ to the concatenation of k vectors $\mathbf{X}_u$ interleaved with k vectors $\vec{\mathbf{0}} \in \mathbb{R}^c$. This is equivalent to the linear transformation given by left multiplication by $(\mathbf{I}_{c\times c}, \mathbf{0}, \ldots, \mathbf{I}_{c\times c}, \mathbf{0})^T \in \mathbb{R}^{2ck\times c}$. After the $2ck$ dimensional BuNN network, we will also need a linear decoder $\text{pool} : \mathbb{R}^{2ck} \to \mathbb{R}^c$ applied to every node individually, which is the sum of the k different c-dimensional vectors that are at even indices. This is equivalent to left multiplication by the matrix $(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}, \ldots, \mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}) \in \mathbb{R}^{c\times 2ck}$. These two can be seen as a linear encoder and linear decoder, often used in practical GNN implementations. We prove the result by adding the lifting and pooling layers and using the higher dimensional $\widehat{BuNN}$ layer, so $\text{BuNN} = \text{pool} \circ \widehat{\text{BuNN}} \circ \text{lift}$.

Defining the 'universal bundle': We fix $k = \sum_{\mathsf{G}\in\mathcal{K}} |\mathsf{V}_\mathsf{G}|$, so we can interpret our embedding space as a lookup table where each index corresponds to a node $v \in \bigsqcup_{\mathsf{G}\in\mathcal{K}} V_\mathsf{G}$. In turn, we can think of the parameter matrix $\mathbf{W} \in \mathbb{R}^{\left(\sum_{G\in\mathcal{K}}|\mathsf{V}_\mathsf{G}|\right)\times\left(\sum_{G\in\mathcal{K}}|\mathsf{V}_\mathsf{G}|\right)}$ as a lookup table where each entry corresponds to a pair of nodes in our dataset $\mathcal{K}$. Still thinking of the indices of the $2ck$ dimensions as $2c$-dimensional vectors indexed by the k nodes in our dataset, we define $\mathbf{O}_u \in O(2ck)$ as a block diagonal matrix with k different $2c$-dimensional blocks. These are all set to the identity except for the block at the index corresponding to node u, which is defined as $\begin{pmatrix} \mathbf{0}_{\mathbf{c}\times\mathbf{c}} & \mathbf{I}_{c\times c} \\ \mathbf{I}_{c\times c} & \mathbf{0}_{\mathbf{c}\times\mathbf{c}} \end{pmatrix}$ which is a $2c \times 2c$ matrix that acts by permuting the first c dimensions with the second c dimensions.

Computing the partial derivatives. Since our model BuNN is a composition of linear maps, and since the maps *pool* and $lift$ are applied node-wise, we get

$$\begin{aligned}
&\frac{\partial \left(\text{BuNN}\left(\mathbf{X}\right)\right)_u}{\partial \mathbf{x}_v} \\
&= \text{pool}\, \frac{\partial \left(\widehat{\text{BuNN}}\left(\text{lift}\left(\mathbf{X}\right)\right)\right)_u}{\partial\, \text{lift}\left(\mathbf{X}_v\right)}\, \text{lift} \\
&= \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}, \ldots, \mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right) \mathcal{H}(t,u,v) \mathbf{O}_u^T \mathbf{W} \mathbf{O}_v \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}, \ldots, \mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right)^T \\
&= \mathcal{H}(t,u,v) \sum_{1\le k_1,\ k_2\le k} \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right) \mathbf{O}_u^T \mathbf{W} \mathbf{O}_v \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right)^T
\end{aligned}$$

We proceed by partitioning the indexing by $(k_1,\ k_2)$ into four cases. The first case is $C_1 = \{(k_1,k_2)$ such that $(k_1 \neq u,v$ and $k_2 \neq u,v)$ or$(k_1 = k_2 = u$ or $k_1 = k_2 = v)\}$ for which both $\mathbf{O}_u$ and $\mathbf{O}_v$ act like the identity. The second case is $C_2 = \{(k_1,k_2)$ such that $k_1 = u$ and $k_2 \neq u,v\}$ where $\mathbf{O}_u$ flips the first c rows with the second c rows and $\mathbf{O}_v$ acts like the identity. $C_3 = \{(k_1,k_2)$ such that $k_2 = j$ and $k_1 \neq u,v\}$ where $\mathbf{O}_v$ flips the first c columns with the second c columns, and $\mathbf{O}_u$ acts like the identity on the rows. Finally, the last case is when $k_1 = u$ and $k_v = v$ in which $\mathbf{O}_v$ flips the rows and $\mathbf{O}_v$ flips the columns.

$$\begin{aligned}
\ldots &= \mathcal{H}(t,u,v) \sum_{1\le k_1,\ k_2\le k} \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right) \mathbf{O}_u^T \mathbf{W} \mathbf{O}_v \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right)^T \\
&= \mathcal{H}(t,u,v) \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right) \left[\sum_{(k_1,k_2)\in C_1} \begin{pmatrix} \mathbf{W}_{00}^{k_1k_2} & \mathbf{W}_{01}^{k_1k_2} \\ \mathbf{W}_{10}^{k_1k_2} & \mathbf{W}_{11}^{k_1k_2} \end{pmatrix} + \sum_{(k_1,k_2)\in C_2} \begin{pmatrix} \mathbf{W}_{10}^{k_1k_2} & \mathbf{W}_{11}^{k_1k_2} \\ \mathbf{W}_{00}^{k_1k_2} & \mathbf{W}_{01}^{k_1k_2} \end{pmatrix} \right. \\
&\qquad \left. + \sum_{(k_1,k_2)\in C_3} \begin{pmatrix} \mathbf{W}_{01}^{k_1k_2} & \mathbf{W}_{00}^{k_1k_2} \\ \mathbf{W}_{11}^{k_1k_2} & \mathbf{W}_{10}^{k_1k_2} \end{pmatrix} + \begin{pmatrix} \mathbf{W}_{11}^{ij} & \mathbf{W}_{10}^{ij} \\ \mathbf{W}_{01}^{ij} & \mathbf{W}_{00}^{ij} \end{pmatrix} \right] \left(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c}\right)^T
\end{aligned}$$

Finally, after applying $(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c})$ on the left and $(\mathbf{I}_{c\times c}, \mathbf{0}_{c\times c})^T$ on the right (an operation that selects the upper left $c \times c$ block), we observe that setting all $\mathbf{W}_{00}^{k_1k_2} = \mathbf{W}_{01}^{k_1k_2} = \mathbf{W}_{10}^{k_1k_2}$ to $\mathbf{0}_{c\times c}$ and setting $\mathbf{W}_{11}^{uv} := \frac{1}{\mathcal{H}(t,u,v)} \frac{\partial (L\mathbf{X})_u}{\partial \mathbf{x}_v}$ if the nodes corresponding to k_1 and k_2 lie in the same graph and $\mathbf{0}_{c\times c}$ otherwise. This allows us to conclude that any linear layer can be parameterized. □

F. Discussion on compact uniform approximation versus uniform approximation

A strong definition of expressivity that deals with infinite collections of graphs was proposed in Rosenbluth et al. (2023). This definition subsumes graph-isomorphism testing (where the input feature on graphs is constant) and is finer than graph-level function approximation since it works at the node level. Furthermore, it also deals with infinite families of graphs, as opposed to most mainstream theorems of GNN expressivity, which are proved for graphs of bounded size (e.g. Azizian & marc lelarge (2021); Geerts & Reutter (2022)) (see Section 2 for the notation and definition of features transformations).

Definition F.1 (From Rosenbluth et al. (2023))**.** Let $c,\ c' \in \mathbb{N}$ and take $\mathbb{R}$ as feature space. Consider a collection of graphs $\mathcal{G}$. Let $\Omega \subseteq \mathcal{C}\left(\mathcal{G}, \mathbb{R}^c, \mathbb{R}^{c'}\right)$ be a set of feature transformations over $\mathcal{G}$, and let $H \in \mathcal{C}\left(\mathcal{G}, \mathbb{R}^c, \mathbb{R}^{c'}\right)$ a feature transformation over $\mathcal{G}$. We say that Ω *uniformly additively approximates* H, notated $\Omega \approx H$ if $\forall \epsilon > 0\ \forall$ compact $K^n \subset \mathbb{R}^{nc}\ \exists F \in \Omega$ such that:, $\forall \mathsf{G} \in \mathcal{G}\ \forall X \in K^{n_{\mathsf{G}}c}\ \left\|F_{\mathsf{G}}(X) - H_{\mathsf{G}}(X)\right\|_\infty \leq \epsilon$ where the sup norm $\|\cdot\|_\infty$ is taken over all nodes and dimensions of $n_{\mathsf{G}}c'$ dimensional output.

Note that this definition differs from our Definition D.1 in that it requires uniform approximation over all graphs in $\mathcal{G}$ simultaneously, while we allow the width to vary with the finite subset $\mathcal{K} \subseteq \mathcal{G}$, similar to how classical results allow the width to vary with the compact set over which to approximate the function. Such a definition has proven useful in Rosenbluth et al. (2023) to distinguish different aggregation functions and in Rosenbluth et al. (2024) to distinguish MPNNs with virtual nodes from Transformers. However, we argue that **the definition above is too strong for a finite parameter GNN**. This is because it requires uniform approximation over a *non-compact set*, which contrasts with traditional work on expressivity and is generally unfeasible and impractical. Indeed, finite-parameters MLPs are not universal over the whole domain $\mathbb{R}$ under the ℓ_∞-norm. On an infinite collection of featured graphs, the topology is the disjoint union topology on $\bigsqcup_{\mathsf{G}\in\mathcal{G}} \mathbb{R}^{n_{\mathsf{G}}d}$, a compact subset consists of a finite set of graphs, and for each graph G only non-zero on a compact subset of $\mathbb{R}^{nd}$. For these reasons, we introduce Definition D.1, which is still rich enough to distinguish between BuNNs and MPNNs.

G. Algorithmic and Implementation details

In this Section, we provide more details on the implementation of BuNNs. We start by discussing how to use several vector-field channels when the input dimension is greater than the bundle dimension. We then discuss how to use several bundles at once when a single bundle is insufficient. We then combine both views, namely having several vector-field channels on several bundles at once. Finally, we describe how we compute our bundle maps in the experiments.

Extending to several vector-field channels. When the signal dimension exceeds the bundle dimension, i.e. $c > d$, we cannot directly apply BuNNs to the input signal. In that case, we first transform the signal into a hidden dimension, a multiple of the bundle dimension, i.e.$c = dp$. We reshape the input signal into p channels of d-dimensional vector fields, where we apply the diffusion step (Equation 3) on each p channels simultaneously, and we apply the weight matrix $\mathbf{W} \in \mathbb{R}^{dp\times dp}$ by first flattening the node signals into dp dimensions, then multiplying by $\mathbf{W}$, and then reshaping it into p channels of d dimensional vector fields.

Extending to several bundles. Learning a high dimensional orthogonal matrix $\mathrm{O}(d)$ becomes expensive since the manifold of orthogonal matrices is $\frac{d(d-1)}{2}$ dimensional. However, we can compute many low-dimensional bundles in parallel. In practice, we found that using several 2-dimensional bundles was enough. Computing b different 2-dimensional bundles requires only b-parameters since the manifold $\mathrm{O}(2)$ is 1-dimensional. We, therefore, also use different 'bundle channels' given by an additional hyper-parameter – the number of bundles, which we denote b. Given an input signal of dimension $c = db$, we can decompose the signal into b bundle channels of dimension d. We can compute the diffusion step (Equation 3) for each bundle in parallel. For the update step (Equation 2), we apply the weight matrix $\mathbf{W} \in \mathbb{R}^{bd\times bd}$ by first flattening the node signals into bd dimensions, then multiplying by $\mathbf{W}$, and then reshaping it into b bundle channels of d dimensional vector fields over b different bundle structures.

Remark G.1. We note that using b different d dimensional bundles is equivalent to parameterizing a subset of one bd-dimensional structure, consisting of the orthogonal map $\mathbf{O} \in O(bd) \subset \mathbb{R}^{bd\times bd}$ that are block diagonal matrices $\mathbf{O} = \bigoplus_{i=1\dots b} \mathbf{O}_i$, with each $\mathbf{O}_i \in O(d)$.

Extending to several bundles and vector-field channels. We can combine the above two observations. Given an input signal of dimension $c = bdp$, we can subdivide this into b different bundle structures of dimension d and p channels for each bundle. We diffuse on the appropriate bundle structure and flatten the vector fields into a $c \times c$ vector before applying the learnable parameters.

Algorithm 1 Taylor expansion implementation of a BuNN layer

Input: Normalized graph Laplacian $\mathcal{L}$, Orthogonal maps $\mathbf{O}_v^{(\ell)}$ $\forall v \in \mathsf{G}$, Node features $\mathbf{X}^{(\ell)} \in \mathbb{R}^{n\times d}$, Time t, Maximum degree K, Channel mixing matrix $\mathbf{W}^{(\ell)}$, bias $\mathbf{b}^{(\ell)}$
Output: Updated node features $\mathbf{X}^{(\ell)}$
$\mathbf{h}_v^{(\ell)} \leftarrow {\mathbf{O}_v^{(\ell)}}^T \mathbf{x}_v^{(\ell)}$ $\forall v \in \mathsf{V}$ {Go to global representation}
$\mathbf{H}^{(\ell)} \leftarrow \mathbf{H}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}$ {Update features with parameters}
$\mathbf{X}^{(\ell+1)} \leftarrow \mathbf{H}^{(\ell)}$ {approximation of degree 0}
for $k = 1, \dots K$ **do**
  $\mathbf{H}^{(\ell)} \leftarrow -\frac{t}{k}\mathcal{L}\mathbf{H}^{(\ell)}$ {Term of degree k}
  $\mathbf{X}^{(\ell+1)} \leftarrow \mathbf{X}^{(\ell+1)} + \mathbf{H}^{(\ell)}$ {Approximation of degree k}
end for
$\mathbf{x}_v^{(\ell+1)} \leftarrow {\mathbf{O}_v^{(\ell)}}^T \mathbf{x}_v^{(\ell+1)}$ $\forall v \in \mathsf{V}$ {Return to local representation}
$\mathbf{X}^{(\ell+1)}$

Computing the bundle maps. In our experiments, we noticed that having several bundles of dimension 2 was more efficient than one bundle of large dimensions, while there was no clear performance gain when using higher dimensional bundles. To compute the b bundle maps $\mathbf{O}_v$ we therefore only need b rotation angles θ_v, one per bundle. In our experiments, we use Housholder reflections using the python package noa or direct parameterization. For direct parameterization, we do the following: since the matrix group $O(2)$ is disconnected, we always take b to be even and parameterize half the bundles as rotation matrices $r(\theta) = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$ and the other half to correspond to matrices with determinant -1, which can be parameterized by $r^*(\theta) = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{pmatrix}$. We compute the angles θ as in Equation 1 where the network $\boldsymbol{\phi}^{(\ell)}$ is either an MLP or a GNN. The network $\boldsymbol{\phi}$ is either shared across layers or differing at every layer.

Taylor approximation algorithm. We now provide pseudo-code on how we implement Equations 2, and 3. We then proceed with a complexity analysis. The key idea of the algorithm is that the bundle heat kernel can be approximated efficiently using the standard graph heat kernel.

The complexity of the algorithms is as follows. There are 3 matrix-vector multiplications done at each node in lines 1, 2, and 8, which are done in $\mathcal{O}\left(3d^2|\mathsf{V}|\right)$. The for loops consist of matrix-matrix multiplications, which are done in $\mathcal{O}\left(d|\mathsf{E}|\right)$ with sparse matrix-vector multiplication. The memory complexity is $\mathcal{O}\left((d+d^2)|\mathsf{V}|\right)$ since we need to store d dimensional vectors and the orthogonal maps for each node. The exact implementation is described in Algorithm 1

Spectral method. We now describe how to implement a BuNN layer using the eigenvectors and eigenvalues of the Laplacian.

Algorithm 2 Spectral implementation of a BuNN layer

1: **Input**: Eigenvectors and eigenvalues graph Laplacian $(\phi_i, \lambda_i)_i$, Orthogonal maps $\mathbf{O}_v^{(\ell)}$ $\forall v \in \mathsf{G}$, Node features $\mathbf{X}^{(\ell)} \in \mathbb{R}^{n\times d}$, Time t, Maximum degree K, Channel mixing matrix $\mathbf{W}^{(\ell)}$, bias $\mathbf{b}^{(\ell)}$
2: **Output**: Updated node features $\mathbf{X}^{(\ell)}$
3: $\mathbf{h}_v^{(\ell)} \leftarrow \mathbf{O}_v^{(\ell)} \mathbf{x}_v^{(\ell)}$ $\forall v \in \mathsf{V}$ {**Sync.**: Go to global representation}
4: $\mathbf{H}^{(\ell)} \leftarrow \mathbf{H}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}$ {Update features with parameters}
5: $\mathbf{X}^{(\ell+1)} \leftarrow \sum_i e^{-t\lambda_i} \phi_i \phi_i^T \mathbf{H}^{(\ell)}$ {Spectral solution to heat equation}
6: $\mathbf{x}_v^{(\ell+1)} \leftarrow {\mathbf{O}_v^{(\ell)}}^T \mathbf{x}_v^{(\ell+1)}$ $\forall v \in \mathsf{V}$ {**Desync.**: Return to local representation}
7: $\mathbf{X}^{(\ell+1)}$

G.1. Householder reflections.

Many different parameterizations of the group $O(n)$ exist. While direct parameterizations are possible for $n = 2,\ 3$ it becomes increasingly complex to do so for larger n, and a general method working for all n is a desirata. While there are several methods to do so, we use Householder reflection since it is used in related methods (Bodnar et al., 2022). We

use the Pytorch package from (**?**). Given given k vectors $v_i \in \mathbb{R}^d$, define the Householder matrices as $H_i = I - 2\frac{v_i v_i^T}{\|v_i\|_2^2}$, and define $U = \prod_{i=1}^k H_i$. All orthogonal matrices may be obtained using the product of d such matrices. Hence the map $\mathbf{R}^{d\times d} \to O(d)$ mapping $V = (v_i)$ to U is a parameterization of the orthogonal group. We use pytorch implementations allowing autograd provided in (Obukhov, 2021).

G.2. BuNN-Hop extension.

In practice we also explored a variation of BuNNs inspired by MixHop (Abu-El-Haija et al., 2019) which we call BuNN-Hop (described in Algorithm 3). We found that a slightly different model performed better on the `Peptides` benchmark datasets. One layer of the model can be derived by running a discrete time diffusion process with update $\mathbf{X}(t) = \mathbf{A}_{\mathcal{B}}\mathbf{X}(t-1)$ and integrating over it using attention term $\boldsymbol{\alpha}(t)$. The resulting update equation is $\mathbf{Y} = \sum_{k=1}^K \boldsymbol{\alpha}_k \mathbf{A}_{\mathcal{B}}^k \mathbf{X}$ where $\boldsymbol{\alpha}$ satisfy $\sum_{k=1}^K \boldsymbol{\alpha}_k = 1$.

Algorithm 3 BuNN-Hop layer implementation.

Input: Normalized graph adjacency $\mathbf{A}$, Orthogonal maps $\mathbf{O}_v^{(\ell)}$ $\forall v \in \mathsf{G}$, Node features $\mathbf{X}^{(\ell)} \in \mathbb{R}^{n\times d}$, Maximum degree K, Channel mixing matrix $\mathbf{W}^{(\ell)}$, bias $\mathbf{b}^{(\ell)}$, attention terms $\boldsymbol{\alpha}^{(\ell)}$
Output: Updated node features $\mathbf{X}^{(\ell)}$
$\mathbf{h}_v^{(\ell)} \leftarrow {\mathbf{O}_v^{(\ell)}}^T \mathbf{x}_v^{(\ell)} \ \ \forall v \in \mathsf{V}$ {Go to global representation}
$\mathbf{H}^{(\ell)} \leftarrow \mathbf{H}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}$ {Update features with parameters}
$\mathbf{X}^{(\ell+1)} \leftarrow \mathbf{H}^{(\ell)}$ {approximation of degree 0}
for $k = 1, \dots K$ **do**
  $\mathbf{H}^{(\ell)} \leftarrow \mathbf{A}\mathbf{H}^{(\ell)}$ {Term of degree k}
  $\mathbf{X}^{(\ell+1)} \leftarrow \mathbf{X}^{(\ell+1)} + \boldsymbol{\alpha}_k^{(\ell)}\mathbf{H}^{(\ell)}$ {Approximation of degree k}
end for
$\mathbf{x}_v^{(\ell+1)} \leftarrow {\mathbf{O}_v^{(\ell)}}^T \mathbf{x}_v^{(\ell+1)} \ \ \forall v \in \mathsf{V}$ {Return to local representation}
$\mathbf{X}^{(\ell+1)}$

H. Experiment details

In this Section we provide additional information about the experiments on the heterophilic graph benchmarks, the LRGB benchmarks, and the synthetic experiments. All experiments were ran on a cluster using NVIDIA A10 (24 GB) GPUs, each experiment using at most 1 GPU. Each machine in the cluster has 64 cores of Intel(R) Xeon(R) Gold 6326 CPU at 2.90GHz, and ∼500GB of RAM available. The synthetic experiments from Section 4.1 were run on CPU and each run took roughly 20 minutes. The heterophilic experiments from Section 4 were run GPU and varied between 5 minutes to 1.5 hours. The LRGB experiments were run on GPU and varied between 0.5 hours and 4 hours.

H.1. LRGB: training and tuning.

For `peptides-func` and `peptides-struct` we use a fixed parameter budget of $\sim 500k$ as in Dwivedi et al. (2022). We fix hyper-parameters to be the best GCN hyper-parameters from Tönshoff et al. (2023), and tune only BuNN-specific parameters as well as the use of BatchNorm. In Table 5, we report the grid of hyper-parameters that we searched, and denote in bold the best combinations of hyper-parameters. The parameters fixed from Tönshoff et al. (2023) are the following:

- Dropout 0.1
- Learning rate 0.001
- Head depth 3
- Positional Encoding: LapPE for `struct` and RWSE for `func`
- Optimizer: AdamW with a cosine annealing learning rate schedule and linear warmup.
- Batch size 200

- Weight decay 0
- We use skip connection as implemented in Dwivedi et al. (2022) and not in Tönshoff et al. (2023). That is, the skip connection does not skip the non-linearity.

For the BuNN specific parameters, we use 2 dimensional bundles, whose angles θ we compute with the help of a small SumGNN architecture using a sum aggregation as defined by $\theta_v^{(\ell)} = \sigma\left(\mathbf{W}_{\text{s}}\mathbf{x}_v^{(\ell)} + \mathbf{W}_{\text{n}}\sum_{u\in\mathcal{N}(v)}\mathbf{x}_u^{(\ell)}\right)$ where the input dimension is the hidden dimension, the hidden dimension is twice the number of bundles and the output is the number of bundles. The number of SumGNN layers is a hyper-parameter we tune. When it is 0 we use a 2 layer MLP with hidden dimension also twice number of bundles. For each hyper-parameter configuration, we set the hidden dimension in order to respect to the parameter budget. We use truncated Taylor series for this implementation.

Parameters	All Values	Best Values			
		BuNN		BuNN-Hop	
		`func`	`struct`	`func`	`struct`
Num bundles b	4, 8, 16	16	16	16	16
Number of BuNN layers	$1-6$	6	4	6	4
Number of SumGNN layer	$0-3$	1	0	6	4
Max expansion degree K	1 2 4 8 16 32 64 128	128	1	64	64
Time t	0.1, 1, 5, 6, 7, 8, 9, 10, ∞	1	6	N/A	N/A

Table 5. Grid of hyper-parameters for `peptides-func` and `peptides-struct`.

H.2. Heterophilic graphs: training and tuning.

For the heterophilic graphs we use the source code from Platonov et al. (2023) in which we add our layer definition. We report all training parameters that we have tuned. Namely, we use GELU activation functions, the Adam optimizer with learning rate 3×10^{-5}, and train all models for 2000 epochs and select the best epoch based on the validation set. To compute the bundle maps, we compute the parameters θ with a GraphSAGE architecture shared across layers (ϕ method = shared) or different at each layer (ϕ method = not shared), with hidden dimension dimension the number of bundles. The number of layers of this GNN is a hyper-parameter we tuned, which when set to 0 we use a 2 layer MLP. For each task we manually tuned parameters, which are subsets of the combinations of parameters in the grid from Table 6. The implementation of the heat kernel used is either truncated Taylor series with degree 8, or the Spectral implementation. We report the best performing combination of parameters in Table 7.

Parameters	All Values
Hidden dim	$256, 512$
Num bundles b	$8, 16, 32, 64, 128, 256$
Bundle dimension d	2
Number of BuNN layers	$1-8$
Number of GNN layer ϕ	$0-8$
Taylor expansion degree K	8
Time t	$0.1, 1, 10, 100$
ϕ method	shared, not shared
Dropout	$0.0,\ 0.2$
Learning rate	$3\times10^{-4}, 3\times10^{-5}$

Table 6. Parameters searched when tuning on the heterophilic graph benchmark datasets.

Parameters	Best Values				
	roman-empire	amazon-ratings	minesweeper	tolokers	questions
Hidden dim	512	512	512	512	256
Num bundles b	64	64	256	256	128
Bundle dim	2	2	2	2	2
Number of BuNN layers	6	2	8	6	6
Number of GNN layer	8	0	8	7	6
Taylor expansion degree K	N/A	N/A	8	8	8
Time t	100	1	1	1	1
ϕ method	not shared	not shared	not shared	not shared	shared
Dropout	0.2	0.2	0.2	0.2	0.2
Learning rate	3×10^{-4}	3×10^{-4}	3×10^{-5}	3×10^{-5}	3×10^{-5}

Table 7. Best parameter for each dataset in the heterophilic graph benchmarks.