Table 5: Dataset Statistics.

Dataset	#Graphs	Average #Nodes	Average #Edges	#Class	Setup		Metric
					Input Level	Task	
Long-range Graph Benchmark [17]							
COCO-SP	123,286	476.9	2693.7	81	Node	Classification	F1 score
PascalVOC-SP	11,355	479.4	2710.5	21	Node	Classification	F1 score
Peptides-Func	15,535	150.9	307.3	10	Graph	Classification	Average Precision
Peptides-Struct	15,535	150.9	307.3	11 (regression)	Graph	Regression	Mean Absolute Error
GNN Benchmark [16]							
MNIST	70,000	70.6	564.5	10	Graph	Classification	Accuracy
CIFAR10	60,000	117.6	941.1	10	Graph	Classification	Accuracy
Pattern	14,000	118.9	3,039.3	2	Node	Classification	Accuracy
MalNet-Tiny	5,000	1,410.3	2,859.9	5	Graph	Classification	Accuracy
Heterophilic Benchmark [58]							
Roman-empire	1	22,662	32,927	18	Node	Classification	Accuracy
Amazon-ratings	1	24,492	93,050	5	Node	Classification	Accuracy
Minesweeper	1	10,000	39,402	2	Node	Classification	ROC AUC
Tolokers	1	11,758	519,000	2	Node	Classification	ROC AUC
Very Large Dataset [30]							
OGBN-Arxiv	1	169,343	1,166,243	40	Node	Classification	Accuracy

A DETAILS OF DATASETS

The statistics of all the datasets are reported in Table 5. For additional details about the datasets, we refer to the Long-range graph benchmark [17], GNN Benchmark [16], Heterophilic Benchmark [58], and Open Graph Benchmark [30].

B DETAILS OF BASELINES

We compare our GMNs with:

(1) MPNNs:
- GCN [33]: GCN is the graph convolutional network proposed by Kipf and Welling [33].
- GIN [80]: GIN is the graph isomorphism network proposed by Xu et al. [80], which is provably as powerful as WL isomorphism test.
- Gated-GCN [10]: Gated-GCN leverages the vanilla graph ConvNet architecture and the edge gating mechanism [48].

(2) Random walk based:
- CRaWl [70]: CraWl is a random walk-based method that uses random walks to construct an encoding matrix for each node, describing the position of the node in the graph, and then employ a 1-d CNN on top of the matrices.

(3) Graph Transformers:
- SAN [35]: SAN is a graph transformer architecture that instead of using pre-defined PE, it uses a learned PE that can take advantage of the full Laplacian spectrum to learn the position of each node in a given graph.
- NAGphormer [11]: NAGphormer is a graph transformer for node classification tasks. For each node, it uses the node's 1-hop, 2-hop, ..., k-hop neighborhoods as its corresponding tokens. It then employs a Transformer module to encode these tokens.
- Graph ViT [28]: Graph ViT is the generalization of Vision Transformers to graphs for graph classification tasks, that uses a graph partitioning algorithm to partition the graph and treat each partition as a graph patch. Then it uses GCN to encode each patch and then apply a Vision Transformer (ViT) on the encoding of graph patches.
- GOAT [34]: GOAT is a graph trasnformer architecture that uses two attention mechanism–local and global attention– to capture both local and global information in the graph.
- GPS [59]: GPS is a general, powerful, and scalable framework that adapts Transformers for graph-structured data.
- Exphormer[65]: Replaces the dense attention mechanism in GPS framework with a new sparse attention module that is inspired by BigBird [85].

(4) Variants of our GMNs:
- GPS + Mamba: This baseline is the case that we use $m = 0$ (node tokenization) with a simple one directional Mamba.
- GMN-: GMN- is GMN model without any PE/SE and MPNN in its architecture framework.

C HYPERPARAMETERS

We use grid search to tune hyperparameters and the search space is reported in Table 6. Following previous studies, we use the same split of traning/test/validation as [59]. We report the results over the 4 random seeds. Also, for the baselines' results (in Tables 1, 2, and 3), we have re-used and reported the benchmark results in the work by Deng et al. [13], Shirzad et al. [65], Tönshoff et al. [69].

Table 6: Search space of hyperparameters for each dataset.

Dataset	M	s	#Layers	Max # Epochs	Learning Rate
Long-range Graph Benchmark [17]					
COCO-SP	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{4, 5\}$	300	0.001
PascalVOC-SP	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{4, 5\}$	300	0.001
Peptides-Func	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{4, 5\}$	300	0.001
Peptides-Struct	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{4, 5\}$	300	0.001
GNN Benchmark [16]					
MNIST	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
CIFAR10	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
Pattern	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
MalNet-Tiny	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
Heterophilic Benchmark [58]					
Roman-empire	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
Amazon-ratings	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
Minesweeper	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
Tolokers	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001
Very Large Dataset [30]					
OGBN-Arxiv	$\{1, 2, 4, 8, 16, 32\}$	$\{0, 1, 2, 4, 8, 16\}$	$\{3, 4, 6\}$	300	0.001

D THEORETICAL ANALYSIS OF GMNS

THEOREM 4. *With large enough $M, m,$ and $s > 0$, GMNs' neighborhood sampling is strictly more expressive than k-hop neighborhood sampling.*

PROOF. We first show that in this condition, the random walk neighborhood sampling is as expressive as k-hop neighborhood sampling. To this end, given an arbitrary small $\epsilon > 0$, we show that the probability that k-hop neighborhood sampling is more expressive than random walk neighborhood sampling is less than ϵ. Let $m = k$, $s = 1$, and $p_{u,v}$ be the probability that we sample node v in a walk with length $m = k$ starting from node u. This prbobality is zero if the shortest path of u and v is more than k. To construct the subgraph token corresponds to $\hat{m} = k$, we use M samples and so the probability of not seeing node v in these samples is $q_{u,v,M} = (1 - p_{u,v})^M \leq 1$. Now let $M \to \infty$ and $v \in \mathcal{N}_k(u)$ (i.e., $p_{u,v} \neq 0$), we have $\lim_{M\to\infty} q_{u,v,M} = 0$. Accordingly, with large enough M, we have $q_{u,v,M} \leq \epsilon$. This means that with a large enough M when $m = k$ and $s = 1$, we sample all the nodes within the k-hop neighborhood, meaning that random walk neighborhood sampling at least provide as much information as k-hop neighborhood sampling with arbitrary large probability.
Next, we provide an example that k-hop neighborhood sampling is not able to distinguish two non-isomorphism graphs, while random walk sampling can. Let $S = \{v_1, v_2, \dots, v_\ell\}$ be a set of nodes such that all nodes have shortest path less than k to u. Using hyperparamters $m = k$ and arbitrary M, let the probability that we get $G[S]$ as the subgraph token be $1 > q_S > 0$. Using s samples, the probability that we do not have $G[S]$ as one of the subgraph tokens is $(1 - q_S)^s$. Now using large $s \to \infty$, we have $\lim_{s\to\infty}(1 - q_S)^s = 0$ and so for any arbitrary $\epsilon > 0$ there is a large $s > 0$ such that we see all non-empty subgraphs of the k-hop neighborhood with probability more than $1 - \epsilon$, which is more powerful than the neighborhood itself.
Note that the above proof does not necessarily guarantee an efficient sampling, but it guarantees the expressive power. □

THEOREM 5 (UNIVERSALITY). *Let $1 \leq p < \infty$, and $\epsilon > 0$. For any continues function $f : [0, 1]^{d\times n} \to \mathbb{R}^{d\times n}$ that is permutation equivariant, there exists a GMN with positional encoding, g_p, such that $\ell^p(f, g) < \epsilon$.*

PROOF. Recently, Wang and Xue [74] show that SSMs with layer-wise nonlinearity are universal approximators of any sequence-to-sequence function. We let $m = 0$, meaning we use node tokenization. Using the universality of SSMs for sequence-to-sequence function, the rest of the proof is the same as Kreuzer et al. [35], where they use the padded adjacency matrix of G as a positional encoding to prove the same theorem for Transformers. In fact, the universality for sequence-to-sequence functions is enough to show the universality on graphs with a strong positional encoding. □

THEOREM 6 (EXPRESSIVE POWER W/ PE). *Given the full set of eigenfunctions and enough parameters, GMNs can distinguish any pair of non-isomorphic graphs and are more powerful than any WL test.*

PROOF. Due to the universality of GMNs in Theorem 5, one can use a GMN with the padded adjacency matrix of G as positional encoding and learn a function that is invariant under node index permutations and maps non-isomorphic graphs to different values. □

THEOREM 7 (EXPRESSIVE POWER W/O PE AND MPNN). *With enough parameter, for every $k \geq 1$ there are graphs that are distinguishable by GMNs, but not by k-WL test, showing that their expressivity power is not bounded by any WL test.*

PROOF. The proof of this theorem directly comes from the recent work of Tönshoff et al. [70], where they prove a similar theorem for CRaWl [70]. That is, using RWF as $\phi(.)$ in GMNs (without MPNN), makes the GMNs as powerful as CRaWl. The reason is CRaWl uses 1-d CNN on top of RWF, while GMNs use Bidirectional Mamba block on top of RWF: using a broadcast SMM, this block becomes similar to 1-d CNN. □

E DETAILS OF GMN ARCHITECTURE: ALGORITHMS

Algorithm 1 shows the forward pass of the Graph Mamba Network with one layer. For each node, GMN first samples M walks with length $\hat{m} = 1, \dots, m$ and constructs its corresponding tokens, each of which as the induced subgraph of M walks with length $\hat{m}$. We repeat this process s times to have longer sequence and more samples from each hierarchy of the neighborhoods. This part of the algorithm, can be computed before the training process and in CPU. Next, GMNs for each node encode its tokens using an encoder $\phi(.)$, which can be an MPNN (e.g., gated-GCN [10]) or RWF encoding (proposed by Tönshoff et al. [70]). We then pass the encodings to a Bidirectional Mamba block, which we describe in Algorithm 2 (This algorithm is simple two Mamba block [22] such that we use one of the backward or forward ordering of inputs for each of them). At the end of line 15, we have the node encodings obtained from subgraph tokenization. Next, we treat each node as a token and pass the encoding to another bidirectional Mamba, with a specific order. We have used degree ordering in our experiments, but there are some other approaches that we have discussed in the main paper.

Algorithm 1 Graph Mamba Networks (with one layer)

```
Input: A graph $G = (V, E)$, input features $\mathbf{X} \in \mathbb{R}^{n\times d}$, ordered array of
    nodes $V = \{v_1, \dots, v_n\}$, and hyperparameters $M, m$, and $s$.
Optional: Matrix $\mathbf{P}$, whose rows are positional/structural encodings corre-
    spond to nodes, and/or a MPNN model $\Psi(.)$.
Output: The updated node encodings $\mathbf{X}_{\text{new}}$.
 2: for $v \in V$ do        ▷ This block can be done before the training.
 3:     for $\hat{m} = 0, \dots, m$ do
 4:         for $\hat{s} = 1, \dots, s$ do
 5:             $T^{\hat{s}}_{\hat{m}}(v) \leftarrow \emptyset$;
 6:             for $\hat{M} = 1, \dots, M$ do
 7:                 $\mathtt{W} \leftarrow$ Sample a random walk with length $\hat{m}$ starting from
    $v$;
 8:                 $T^{\hat{s}}_{\hat{m}}(v) \leftarrow T^{\hat{s}}_{\hat{m}}(v) \cup \{u | u \in \mathtt{W}\}$;
 9:
10: ▷ Start the training:
11: Initialize all learnable parameters;
12: for $v \in V$ do
13:     for $j = 1, \dots, s$ do
14:         for $i = 1, \dots, m$ do
15:             $\mathbf{x}_v^{(i-1)s+j} \leftarrow \phi\left(G[T^j_i(v)], \mathbf{X}_{T^j_i(v)} \,||\, \mathbf{P}_{T^j_i(v)}\right)$;
        $\Phi_v \leftarrow ||_{i=1}^{sm} \mathbf{x}_v^i$;        ▷ $\Phi_v$ is a matrix whose rows are $\mathbf{x}_v^i$.
16:     $\boldsymbol{y}_{\text{output}}(v) \leftarrow \mathtt{BiMamba}(\Phi_v)$;        ▷ Using Algorithm 2.
17: ▷ Each node is a token:
18: $Y \leftarrow ||_{i=1}^{sm} \boldsymbol{y}_{\text{output}}(v)$;    ▷ $\boldsymbol{y}$ is a matrix whose rows are $\boldsymbol{y}_{\text{output}}(v)$.
19: $Y_{\text{output}} \leftarrow \mathtt{BiMamba}(Y) + \Psi(G, \mathbf{X}||\mathbf{P})$;
```

Algorithm 2 Bidirectional Mamba

```
Input: A sequence $\Phi$ (Ordered matrix, where each row is a token).
Output: The updated sequence encodings $\Phi$.
 1: ▷ Forward Scan:
 2: $\Phi_f = \sigma(\mathtt{Conv}(\mathbf{W}_{\text{input},f}\ \mathtt{LayerNorm}(\Phi)))$;
 3: $\mathbf{B}_f = \mathbf{W}_{\mathbf{B}_f}\ \Phi_f$;
 4: $\mathbf{C}_f = \mathbf{W}_{\mathbf{C}_f}\ \Phi_f$;
 5: $\Delta_f = \mathtt{Softplus}(\mathbf{W}_{\Delta_f}\ \Phi_f)$;
 6: $\bar{\mathbf{A}} = \mathtt{Discrete}_{\mathbf{A}}(\mathbf{A}, \Delta)$;
 7: $\bar{\mathbf{B}}_f = \mathtt{Discrete}_{\mathbf{B}_f}(\mathbf{B}_f, \Delta)$;
 8: $\boldsymbol{y}_f = \mathtt{SSM}_{\bar{\mathbf{A}}, \bar{\mathbf{B}}_f, \mathbf{C}_f}(\Phi_f)$;
 9: $Y_f = \mathbf{W}_{f,1}(\boldsymbol{y}_f \odot \sigma(\mathbf{W}_{f,2}\ \mathtt{LayerNorm}(\Phi)))$;
10: ▷ Backward Scan:
11: $\Phi \leftarrow \mathtt{Reverse\text{-}rows}(\Phi)$;        ▷ Reverse the order of rows in the
    matrix.
12: $\Phi_b = \sigma(\mathtt{Conv}(\mathbf{W}_{\text{input, b}}\ \mathtt{LayerNorm}(\Phi)))$;
13: $\mathbf{B}_b = \mathbf{W}_{\mathbf{B}_b}\ \Phi_b$;
14: $\mathbf{C}_b = \mathbf{W}_{\mathbf{C}_b}\ \Phi_b$;
15: $\Delta_b = \mathtt{Softplus}(\mathbf{W}_{\Delta_b}\ \Phi_b)$;
16: $\bar{\mathbf{A}} = \mathtt{Discrete}_{\mathbf{A}}(\mathbf{A}, \Delta)$;
17: $\bar{\mathbf{B}}_b = \mathtt{Discrete}_{\mathbf{B}_b}(\mathbf{B}_b, \Delta)$;
18: $\boldsymbol{y}_b = \mathtt{SSM}_{\bar{\mathbf{A}}, \bar{\mathbf{B}}_b, \mathbf{C}_b}(\Phi_b)$;
19: $Y_b = \mathbf{W}_{b,1}(\boldsymbol{y}_b \odot \sigma(\mathbf{W}_{b,2}\ \mathtt{LayerNorm}(\Phi)))$;
20: ▷ Output:
21: $\boldsymbol{y}_{\text{output}} \leftarrow \mathbf{W}_{\text{out}}(Y_f + \mathtt{Reverse\text{-}row}(Y_b))$;
22: return $\boldsymbol{y}_{\text{output}}$;
```

F ADDITIONAL EXPERIMENTAL RESULTS

F.1 Parameter Sensitivity

The effect of M. Parameter M is the number of walks that we aggregate to construct a subgraph token. To evaluate its effect on the performance of the GMN, we use two datasets of Roman-empire and PascalVOC-SP, from two different benchmarks, and vary the value of M from 1 to 10. The results are reported in Figure 3 (Left). These results show that performance peaks at certain value of M and the exact value varies with the dataset. The main reason is, parameter M determines that how many walks can be a good representative of the neighborhood of a node, and so depends on the density, homophily score, and network topology this value can be different.

The effect of m. Similar to the above, we use two datasets of Roman-empire and PascalVOC-SP, from two different benchmarks, and vary the value of m from 1 to 60. The results are reported in Figure 3 (Middle). The performance is non-decreasing with respect to the value of m. That is, increasing the value of m, i.e., considering far neighbors in the tokenization process, does not damage the performance (might lead to better results). Intuitively, using large values of m is expected to damage the performance due to the over-smoothing and over-squashing; however, the selection mechanism in Bidirectional Mamba can select informative tokens (i.e., neighborhood), filtering information that cause performance damage.

The effect of s. Using the same setting as the above, we report the results when varying the value of s in Figure 3 (Right). Result show that increasing the value of s can monotonically improve the performance. As discussed earlier, longer sequences of tokens can provide more context for our model and due to the selection mechanism in Mamba [22], GMNs can select informative subgraphs/nodes and filter irrelevant tokens, resulting in better results with longer sequences.

F.2 Efficiency

As we discussed earlier, one of the main advantages of our model is its efficiency and memory usage. We evaluate this claim on OGBN-Arxiv [30] and MalNet-Tiny [16] datasets and report the results in Figure 4. Our variants of GMNs are the most efficienct methods while achieving the best performance. To show the trend of scalability, we use MalNet-Tiny and plot the memory usage of GPS and GMN in Figure 5. While GPS, as a graph transformer framework, requires high computational cost (GPU memory usage), GMNs's memory scales linearly with respect to the input size.

F.3 Full Ablation Study

To evaluate the contribution of each component of GMNs in its performance, we perform ablation study on 9 datasets (three datasets from each benchmark). Table 8 reports the results for all these 9 datasets. The first row, reports the performance of GMNs with its full architecture. Then in each row, we modify one the elements while keeping the other unchanged: Row 2 remove the bidirectional Mamba and use a simple Mamba. Row 3 remove the MPNN and Row 4 use PPR ordering. Finally the last row remove PE. Results

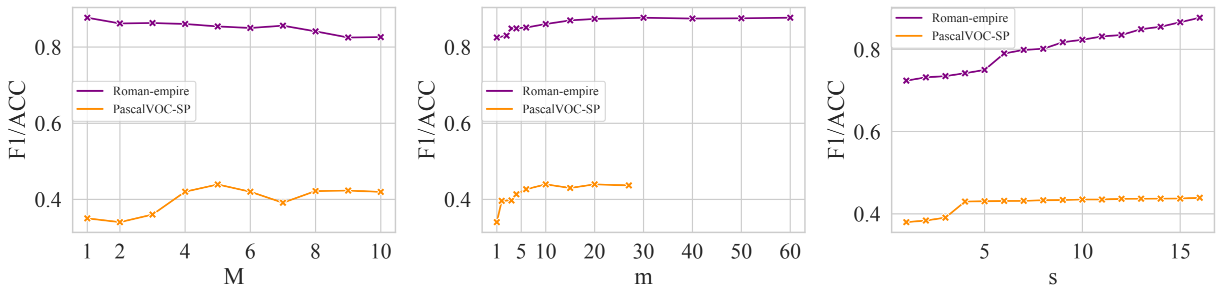

Figure 3: The effect of (Left) M, (Middle) m, and (Right) s on the performance of GMNs.

Figure 4: Efficiency evaluation and accuracy of GMNs and baselines on OBGN-Arxiv and MalNet-Tiny. Highlighted are the top first, second, and third results. OOM: Out of Memory.

Method	Gated-GCN	GPS	NAGphormer	Exphormer[†]	GOAT	Ours: GPS+Mamba	Ours: GMN
OGBN-Arxiv							
Training/Epoch (s)	**0.68**	OOM	5.06	1.97	13.09	**1.18**	**1.30**
Memory (GB)	11.09	OOM	**6.24**	36.18	8.41	**5.02**	**3.85**
Accuracy	0.7141	OOM	0.7013	**0.7228**	0.7196	**0.7239**	**0.7248**
MalNet-Tiny							
Training/Epoch (s)	**10.3**	148.99	-	57.24	-	**36.07**	**41.00**
Accuracy	0.9223	**0.9234**	-	0.9224	-	**0.9311**	**0.9415**

[†] We follow the original paper [65] and use one virtual node in efficiency evaluation.

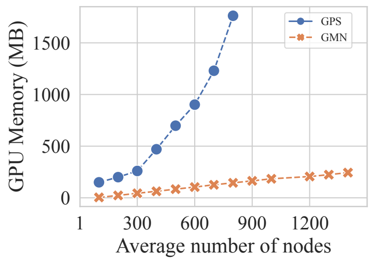

Figure 5: Memory of GPS and GMN on MalNet-Tiny dataset.

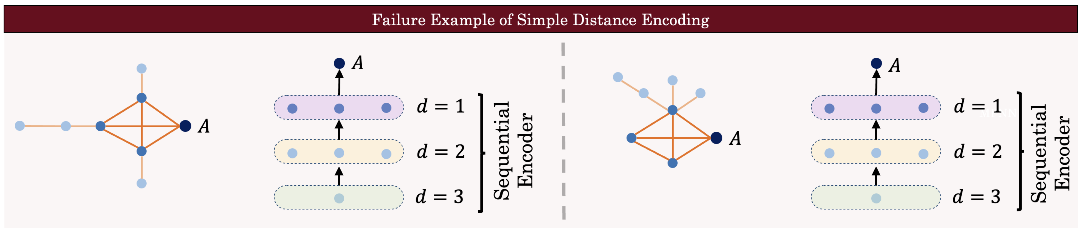

Figure 6: Failure example for methods that are solely based on distance encoding. Solely considering the set of nodes in different distances to the target node misses the connections between them. While the structure of these two graphs are different, the set of nodes with the same distance to node A are the same. Accordingly, they achieve the same node encoding for A, missing A's neighborhood topology.

Table 7: Comparison with GRED and S4G Models. Highlighted are the top first and second results.

Model	MNIST Accuracy ↑	CIFAR10 Accuracy ↑	PATTERN Accuracy ↑	Peptides-func AP ↑	Peptides-struct MAE ↓
S4G [67][†]	$0.9637_{\pm 0.0017}$	$0.7065_{\pm 0.0033}$	$0.8687_{\pm 0.0002}$	$0.7293_{\pm 0.0004}$	$0.2485_{\pm 0.0017}$
GRED [15][‡]	$0.9822_{\pm 0.0095}$	$0.7537_{\pm 0.6210}$	$0.8676_{\pm 0.0200}$	$0.7041_{\pm 0.0049}$	$0.2503_{\pm 0.0019}$
GMN (Ours)	$0.9839_{\pm 0.0018}$	$0.7576_{\pm 0.0042}$	$0.8714_{\pm 0.0012}$	$0.7071_{\pm 0.0083}$	$0.2473_{\pm 0.0025}$

[†] Results are reported by Song et al. [67]. [‡] Results are reported by Ding et al. [15].

show that all the elements of GMN contributes to its performance with most contribution from bidirection Mamba.

F.4 Comparison with GRED [15] and S4G [67]

GRED [15] is a recent work on ArXiv that uses an RNN on the set of neighbors with distance $k = 1, \dots, K$ to a node of interest for the node classification task. S4G [67] is a recent unpublished (without preprint but available on Openreview) work that uses the same approach as GRED but replaces the RNN block with a structured SSM. Since the code or models of GRED and S4G are not available, for the comparison of GMNs, S4G, and GRED, we run GMNs on the datasets used in the original papers [15, 67]. The results are reported in Table 7. GMNs consistently outperforms GRED [15] in all datasets and outperforms S4G in 4 out of 5 datasets. The reason is two folds: (1) GMNs use sampled walks instead of all the nodes within k-distance neighborhood. As discussed in Theorem 4, this

Table 8: Ablation study on GMN architecture.

Model	Roman-empire Accuracy ↑	Amazon-ratings Accuracy ↑	Minesweeper ROC AUC ↑	MNIST Accuracy ↑	CIFAR10 Accuracy ↑	PATTERN Accuracy ↑	PascalVOC-SP F1 Score ↑	COCO-SP F1 Score ↑	Peptides-Func AP ↑
GMN	$0.8769_{\pm 0.0050}$	$0.5407_{\pm 0.0031}$	$0.9101_{\pm 0.0023}$	$0.9839_{\pm 0.0018}$	$0.7576_{\pm 0.0042}$	$0.8714_{\pm 0.0012}$	$0.4393_{\pm 0.0112}$	$0.3974_{\pm 0.0101}$	$0.7071_{\pm 0.0083}$
w/o bidirectional Mamba	$0.8327_{\pm 0.0062}$	$0.5016_{\pm 0.0045}$	$0.8597_{\pm 0.0028}$	$0.9410_{\pm 0.0025}$	$0.7269_{\pm 0.0038}$	$0.8297_{\pm 0.0027}$	$0.4207_{\pm 0.0094}$	$0.3658_{\pm 0.0077}$	$0.6749_{\pm 0.0058}$
w/o MPNN	$0.8620_{\pm 0.0043}$	$0.5312_{\pm 0.0044}$	$0.8983_{\pm 0.0031}$	$0.9693_{\pm 0.0021}$	$0.7458_{\pm 0.0032}$	$0.8588_{\pm 0.0019}$	$0.4217_{\pm 0.0095}$	$0.3810_{\pm 0.0074}$	$0.6894_{\pm 0.0075}$
PPR ordering	$0.8612_{\pm 0.0019}$	$0.5299_{\pm 0.0037}$	$0.8991_{\pm 0.0021}$	$0.9712_{\pm 0.0031}$	$0.7462_{\pm 0.0027}$	$0.8609_{\pm 0.0020}$	$0.4276_{\pm 0.0103}$	$0.3861_{\pm 0.0068}$	$0.6914_{\pm 0.0081}$
w/o PE	$0.8591_{\pm 0.0054}$	$0.5308_{\pm 0.0026}$	$0.9011_{\pm 0.0025}$	$0.9645_{\pm 0.0022}$	$0.7308_{\pm 0.0028}$	$0.8531_{\pm 0.0018}$	$0.4195_{\pm 0.0091}$	$0.3706_{\pm 0.0083}$	$0.6863_{\pm 0.0074}$

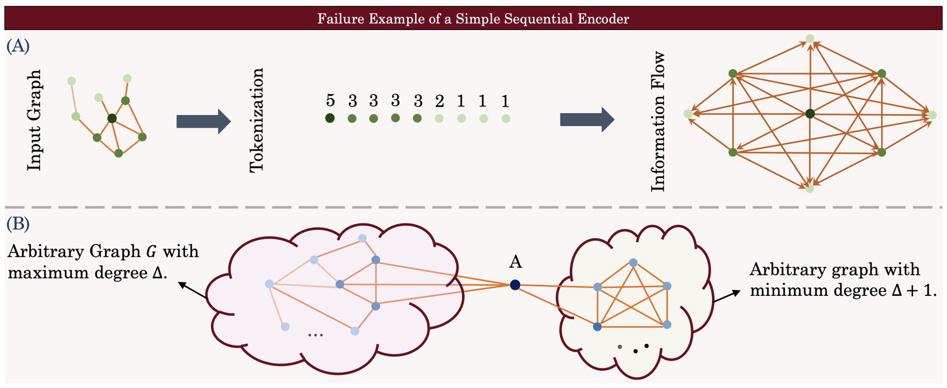
Failure Example of a Simple Sequential Encoder
(A)
Input Graph
Tokenization
5 3 3 3 3 2 1 1 1
Information Flow
(B)
Arbitrary Graph G with maximum degree Δ.
A
Arbitrary graph with minimum degree $\Delta + 1$.

Figure 7: (A) An example of node tokenization and its information flow when using one directional Mamba. Even using PE/SE, nodes at the beginning of the sequence do not have any information about the structure of the graph. (B) Potential failure example for using a simple one directional sequential encoder when each token is a node. Nodes in the right hand side do not have any information about the structure of the graph in the left hand side.

approach with large enough length and samples is more expressive than considering all nodes within the neighborhood. (2) S4G and GRED use simple RNN and SSM to aggregate the information about all the different neighborhoods of a node, while GMNs use Mamba, which have a selection mechanism. This selection mechanism help the model to choose neighborhoods that are more informative and important than others. (3) GRED and S4G are solely based on distance encoding, meaning that they miss the connections between nodes in k-distance and $(k+1)$-distance. Figure 6 shows a failure example of these methods that solely are based on distance of nodes. To obtain the node encoding of node A, these two methods group nodes wit respect to their distance to A, either $d = 1, 2,$ and 3. In Figure 6, while these two graphs are non-isomorphism, the output of this step for both graphs are the same, meaning that these methods obtain the same node encoding for A.

G COMPLEXITY ANALYSIS OF GMNS

$m \geq 1$. For each node $v \in V$, we generate $M \times s$ walks with length $\hat{m} = 1, \ldots, m$, which requires $\mathcal{O}(M \times s \times (m+1))$ time. Given K tokens, the complexity of bidirectional Mamba is 2× of Mamba [22], which is linear with respect to K. Accordingly, since we have $\mathcal{O}(M \times s \times m)$ tokens, the final complexity for a given node $v \in V$ is $\mathcal{O}(M \times s \times (m+1))$. Repeating the process for all nodes, the time complexity is $\mathcal{O}(M \times s \times (m+1) \times |V| + |E|)$, which is linear in terms of $|V|$ and $|E|$ (graph size). To compare to the quadratic time complexity of GTs, even for small networks, note that in practice, $M \times s \times (m+1) \ll |V|$, and in our experiments usually $M \times s \times (m+1) \leq 200$. Also, note that using MPNN as an optional step cannot affect the time complexity as the MPNN requires $\mathcal{O}(|V| + |E|)$ time.

$m = 0$. In this case, each node is a token and so the GMN requires $\mathcal{O}(|V|)$ time. Using MPNN in the architecture, the time complexity would be $\mathcal{O}(|V| + |E|)$, dominating by the MPNN time complexity. As discussed above, based on the properties of Mamba architecture, longer sequence of tokens (larger value of $s \geq 1$) can improve the performance of the method. Based on the abovementioned time complexity when $m \geq 1$, we can see that there is a trade-off between time complexity and the performance of the model. That is, while larger s result in better performance, it results in slower model.

H WHY BIDIRECTIONAL MAMBA?

Using a simple one directional Mamba causes the lack of inductive bias about some structures in the graph. Figure 7 provides an example of this information loss. In part (A), we show an example of node tokenization and node ordering with respect to their degrees. Based on the information flow, using a one directional Mamba, nodes with high-degree do not have any information about the structure of the graph. For example, in Figure 7 (B), even with using complex PE/SE, nodes in the right hand side do not have any information about the global information in the left hand side, due to the one directional information flow of Mamba block. As discussed earlier in the paper, this is one of the new challenges of using SSMs (compare to GTs). The main reason is, attentions in Transformers consider all nodes connected and so the information could pass between each pair of nodes. In sequential encoders (even with selection mechanism), however, each token has the information about its previous tokens.

Our neighborhood sampling and its reverse ordering can address this issue due to its implicit order of neighborhood hierarchy. Also, when using nodes as tokens (either in the last layers or when $m = 0$) our bidirectional Mamba ensure information passes between each pair of nodes.