GNNM: PARAMETER-EFFICIENT BATCHENSEMBLES FOR GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Ensembles are an effective way to improve the predictive performance of neural networks, but their computational and memory costs limit their use in many settings. Modern parameter-efficient ensembles address this issue by sharing the main model weights across ensemble members and introducing only a small number of additional parameters, yet their effectiveness in message passing architectures remains unclear. We introduce GNNM, a lightweight ensemble modification for residual GNNs that replaces only the input and output linear projectors with BatchEnsemble parameterisations while leaving the graph backbone unchanged. This design allows the same modification to be applied to different GNN models with minimal parameter overhead and without altering their message passing layers. We evaluate GNNM on five standard node classification benchmarks and compare it with both individual model baselines and explicit ensemble baselines under the same protocol for hyperparameter selection. The results show that a GNNM variant achieves the best score on every dataset. On average, GNNM improves over the individual model for every backbone and performs on par with or better than an explicit ensemble of the same size while using about four times fewer parameters, while its training cost is comparable to that of the explicit ensemble. The gains are strongest for SAGE, GAT-sep, GT, and GT-sep, while for GAT the explicit ensemble remains stronger. These results show that projector level ensembling can serve as a simple and parameter-efficient mechanism for improving residual GNNs while preserving the structure of the underlying graph model.

1 INTRODUCTION

Ensembling is a general architectural mechanism for improving neural networks. Instead of relying on a single predictor, an ensemble combines several predictors and uses their aggregated output as the final prediction. In practice, this often improves accuracy, calibration, and robustness compared to individual models (Lakshminarayanan et al., 2017). For this reason, ensembles are commonly treated as a strong default whenever the goal is to push predictive quality beyond what can be obtained by a single architecture.

However, the standard explicit ensemble construction can be suboptimal. The usual recipe trains several independent copies of the same model and averages their predictions. This increases model capacity only by duplicating the full architecture, without forcing the ensemble members to share useful structure or interact during training. Recent results in tabular deep learning show that this is not always the best way to use ensembling. BatchEnsemble introduces a shared weight matrix together with lightweight instance-specific multiplicative factors (Wen et al., 2020). TabM builds on this idea and shows that parameter-efficient ensembling can outperform not only strong single-model baselines, but also ordinary explicit ensembles and gradient boosting methods on tabular benchmarks (Gorishniy et al., 2025). This suggests that an ensemble can be more than a collection of independently trained models. Instead, it can be designed as a single architecture in which shared parameters and instance-specific modulations produce a stronger predictor.

In this work, we study whether this effect transfers to graph neural networks. Graph learning differs substantially from tabular prediction. In a multilayer perceptron, samples are processed independently. In a GNN, node representations are repeatedly mixed through message passing, which

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

makes ensemble instances interact with graph structure in a way that has no direct analogue in tabular MLPs. This coupling can reduce the diversity between ensemble instances, which is one of the mechanisms that makes ensembles effective. At the same time, directly stacking several complete GNNs duplicates the entire message-passing backbone. This gives an explicit ensemble, but not necessarily an optimal ensemble architecture.

We introduce GNNM (**GNN** that makes **M**ultiple predictions), a BatchEnsemble modification for GNN pipelines. The method replaces only the input and output linear projectors with BatchEnsemble parameterizations and leaves all message-passing layers unchanged. The input projector controls how node features enter the hidden space, while the output projector maps final node representations to class logits. As a result, all ensemble instances share the graph backbone but use instance-specific feature injection and readout transformations. Since the message-passing operator is left unchanged, the method is not tied to a specific GNN layer and can be applied to different graph backbones without changing their propagation rules.

We evaluate GNNM on standard node classification benchmarks against vanilla GNNs and explicit ensembles built from the same backbones. The results show that projector-level BatchEnsemble is a parameter-efficient alternative to model stacking. On average across datasets and backbones, GNNM improves over the corresponding vanilla models and performs on par with or better than explicit ensembles while keeping the message-passing stack shared. The advantage is in the number of parameters, not in computation: every instance still passes through the shared backbone, so the training step time of GNNM is close to that of an explicit ensemble with the same number of members.

Our contributions are summarized as follows. First, we propose GNNM, a projector-level BatchEnsemble architecture for GNN pipelines that keeps the message-passing backbone shared across ensemble instances. Second, we empirically investigate whether parameter-efficient ensembling can provide advantages beyond conventional explicit ensembling when transferred from tabular architectures to graph neural networks. Third, we compare GNNM against explicit ensembles built from the same GNN backbones and show that, on average, parameter-efficient ensembling matches or exceeds ordinary model stacking at about a quarter of its parameter count, and we support this with significance tests that account for the dependence between results on the same dataset and backbone. Finally, we provide ablations on the ensemble size and initialization strategy.

2 Related Work

2.1 Graph neural networks

Graph neural networks have become a standard approach for learning on graph-structured data. In node classification, GNNs update node representations by aggregating information from neighboring vertices and combining it with the node's current hidden state. Canonical examples include Graph Convolutional Networks (GCN) for semi-supervised node classification (Kipf & Welling, 2017), GraphSAGE for inductive neighborhood aggregation (Hamilton et al., 2017), Graph Attention Networks (GAT) with learned attention weights over neighbors (Veličković et al., 2018), and Graph Isomorphism Networks (GIN), which analyze the expressive power of message passing (Xu et al., 2019).

Despite their success, GNNs are sensitive to architectural and evaluation choices. Increasing depth may lead to oversmoothing, where node representations become less distinguishable after repeated propagation (Rusch et al., 2023). Performance may also depend strongly on the graph homophily level, dataset split, and benchmark protocol (Shchur et al., 2018; Platonov et al., 2023). Consequently, graph learning provides a rigorous evaluation setting for architectural modifications, since a meaningful improvement should not merely increase model capacity but should also remain robust across different backbones and datasets.

Our work is orthogonal to the design of a particular message-passing operator. Instead of introducing a new graph convolutional layer, we modify the linear projectors around the graph backbone. This allows the same ensembling mechanism to be applied to different GNN architectures without changing their propagation rules.

2.2 DEEP AND EFFICIENT ENSEMBLING

Deep ensembles are a strong and widely used method for improving predictive performance and uncertainty estimation. The standard approach trains multiple independently initialized neural networks and averages their predictions (Lakshminarayanan et al., 2017). Although this construction is simple and effective, it treats ensembling as external model replication, because each ensemble member has its own full set of parameters and is trained as a separate model.

Several alternatives aim to obtain ensemble-like benefits without training many fully independent networks. Snapshot ensembles construct an ensemble from a single optimization trajectory by saving model parameters at multiple points during training (Huang et al., 2017). Other approaches use the geometry of the loss landscape to identify low-loss connections between independently trained solutions, thereby enabling more efficient model averaging and ensembling (Garipov et al., 2018). Although these methods reduce the training cost associated with explicit ensembles, they still obtain diversity by combining predictors or parameter states corresponding to distinct points in the optimization landscape. Implicit ensembles instead place several members inside one network, for example through independent subnetworks with multiple inputs and outputs (Havasi et al., 2021), rank-one Bayesian factors (Dusenberry et al., 2020), or grouped Packed-Ensembles (Laurent et al., 2023).

Our focus is different. We treat efficient ensembling as an architectural principle rather than only a training or checkpointing trick. The goal is to build a single GNN pipeline in which shared parameters and instance-specific modulations jointly form an ensemble that can compete with ordinary model stacking.

2.3 ENSEMBLES FOR GRAPH NEURAL NETWORKS

Explicit GNN ensembles are standard baselines for node-level uncertainty (Stadler et al., 2021), and recent work studies how to diversify their members (Deng et al., 2025). Earlier efficient GNN ensembles suppress messages from likely misclassified nodes (Nagarajan et al., 2022) or distill several GNNs into one MLP (Zhang et al., 2024), but still train fully separate members. The closest idea to ours is the shallow ensemble of Vinchurkar et al. (2025), which shares a GNN backbone and ensembles only the output layer for molecular regression. GNNM also shares the backbone, but it uses BatchEnsemble modulations in both the input and the output projector and targets node classification. To our knowledge, BatchEnsemble has not previously been evaluated as a drop-in wrapper for different message-passing backbones. Calibration of GNNs is usually addressed post hoc (Wang et al., 2021; Hsu et al., 2022), and uncertainty estimation is known to be harder on heterophilous graphs (Fuchsgruber et al., 2025).

2.4 BATCHENSEMBLE AND TABM

BatchEnsemble provides a parameter-efficient way to construct ensembles inside a neural network (Wen et al., 2020). Instead of assigning each ensemble member an independent weight matrix, BatchEnsemble uses a shared matrix together with lightweight instance-specific multiplicative factors. For a linear transformation with shared weight matrix W, an ensemble member m applies a transformation of the form

$$\mathrm{BE}_m(X) = ((X \odot r_m)W) \odot s_m + b_m,$$

where r_m, s_m, and b_m are member-specific parameters and $\odot$ denotes element-wise multiplication. This structure keeps most parameters shared while allowing each ensemble member to implement a different transformation.

TabM applies this idea to tabular deep learning and shows that BatchEnsemble-style architectures can outperform not only single neural networks, but also ordinary explicit ensembles and strong gradient boosting baselines on tabular benchmarks (Gorishniy et al., 2025). This result is important for our motivation because it suggests that parameter-efficient ensembling is not merely a cheaper approximation of full ensembles. It can define a stronger architecture in which shared weights and instance-specific factors interact more effectively than independently trained model copies.

Initialization is also important for this class of models. In our GNN adaptation, we follow a TabM-style initialization designed to make the BatchEnsemble parameterization start close to a standard

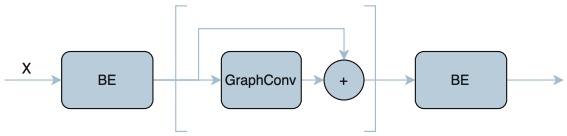

Figure 1: GNNM architecture. BatchEnsemble is applied only to the input and output projectors, while the message-passing backbone is shared across prediction instances.

non-ensembled network. The shared matrices W are initialized with Xavier-uniform initialization. For the first BatchEnsemble projector, the input modulation factors r_m are sampled from $\{-1, +1\}$, while s_m is initialized to one and b_m to zero. For the output BatchEnsemble projector, we use identity-like initialization with $r_m = 1$, $s_m = 1$, and $b_m = 0$. This initialization preserves the scale of the underlying network at the beginning of training while still allowing the first projector to introduce diversity between ensemble instances.

GNNM transfers this BatchEnsemble principle from tabular architectures to graph pipelines. The key difference is that the message-passing backbone remains shared, and BatchEnsemble is applied only to the input and output projectors. This makes the method compatible with standard GNN backbones while aiming to reproduce the ability of TabM-style parameter-efficient ensembling to compete with conventional explicit ensembles.

3 METHOD

We view a standard GNN pipeline as an input projector followed by a message-passing backbone and an output projector.

The specific form of the message-passing layers depends on the chosen GNN operator. GNNM does not require a particular propagation rule. It only assumes that the pipeline contains input and output projectors around the message-passing backbone. This modular view separates feature injection, graph propagation, and prediction readout.

GNNM is a projector-level BatchEnsemble architecture for graph neural networks. Instead of training several independent GNNs, it keeps the message-passing backbone shared and applies BatchEnsemble only to the input and output projectors.

As illustrated in Figure 1, for each BatchEnsemble instance, the input projector maps node features into a separate hidden representation, the shared GNN backbone performs message passing, and the output projector maps the resulting node representations to logits. Thus, different instances use different feature injection and readout transformations, but they rely on the same graph propagation mechanism.

This construction differs from an explicit ensemble. In an explicit ensemble, each member has its own full GNN, including its own message-passing layers. In GNNM, the message-passing backbone is shared across all instances, and ensemble diversity is introduced only through the BatchEnsemble factors in the projector layers. Therefore, GNNM is not a new message-passing operator, but a wrapper that can be applied to an existing GNN backbone without changing its propagation rule.

GNNM produces M logit matrices $Z_1, \ldots, Z_M$, one for each BatchEnsemble instance. At inference time, the final prediction is obtained by averaging the raw instance logits, $Z = \frac{1}{M} \sum_{m=1}^{M} Z_m$, while the explicit ensemble baseline aggregates its members in the same way. All BatchEnsemble instances are trained jointly as a single model. We use the average of the instance-wise losses over the labeled training nodes (cross-entropy for multiclass datasets and binary cross-entropy with logits for binary datasets). In our implementation, the instances are processed one after another through the shared backbone within each training step, and their gradients are accumulated before a single optimizer update.

Compared to a single GNN, GNNM produces multiple predictions through instance-specific projectors. Compared to an explicit ensemble, these predictions are produced without duplicating the message-passing backbone.

4 EXPERIMENTS

4.1 DATASETS

We evaluate GNNM on five standard node classification benchmarks adopted from a recent benchmark suite for heterophilous graph learning (Platonov et al., 2023): Roman-empire, Amazon-ratings, Minesweeper, Tolokers, and Questions. The benchmark contains graphs with different sizes, feature dimensions, numbers of classes, and structural properties, which allows us to evaluate the proposed projector-level BatchEnsemble modification across diverse node classification settings.

We follow the original data partitioning and evaluation setup for each dataset. Model selection is performed using validation performance, after which the selected configuration is evaluated on the corresponding test split. The reported results use the dataset-specific metrics defined by the benchmark.

Additionally, we evaluated GNNM on one graph-classification problem (PROTEINS), where GNNM obtains the highest mean accuracy, although the differences between variants are within the fold-to-fold standard deviation (Table 9 and Appendix D).

4.2 BACKBONE ARCHITECTURES

We use the same set of backbone architectures as the original benchmark study on the five heterophilous node classification datasets (Platonov et al., 2023). Specifically, we consider GCN, GraphSAGE, GAT, GAT-sep, Graph Transformer (GT), GT-sep, and ResNet backbones. In each case, we keep the backbone architecture unchanged relative to the original setup and apply GNNM only as a projector-level modification. Thus, the input and output projectors are replaced with BatchEnsemble parameterizations, while the message-passing layers and the overall backbone design remain unchanged.

For all backbone architectures, we follow the hyperparameter setting of the original benchmark protocol (Appendix A). This includes the same search over 1 to 5 message-passing layers, with the remaining hyperparameters kept fixed as in the original setup. The number of layers is selected on the validation set of the first split and reused for the remaining splits. We use this configuration space and selection procedure for vanilla models, explicit ensembles, and GNNM variants, so the comparison isolates the effect of projector-level BatchEnsemble.

4.3 BASELINES

We compare GNNM against two groups of baselines. The first group consists of vanilla GNNs, where each backbone is trained as a single model without ensembling. This comparison measures whether projector-level BatchEnsemble improves the underlying architecture.

The second group consists of explicit ensembles built from the same GNN backbones. In this setting, several independently initialized copies of the same model are trained, and their logits are averaged at inference time. This comparison evaluates whether GNNM can match or outperform ordinary model stacking rather than only approximate it with fewer parameters.

All comparisons are performed within the same backbone family. A GNNM version of a given backbone is compared with the corresponding vanilla model and with an explicit ensemble of the same backbone.

5 RESULTS

We present the empirical evaluation of GNNM in three parts. First, we compare GNNM with vanilla GNNs and explicit ensembles on the full benchmark protocol. Then, we analyze the effect of the number of BatchEnsemble instances and the initialization strategy in a simplified ablation setup. Finally, we report the computational cost of the proposed architecture.

Variant	Roman-empire	Amazon-ratings	Minesweeper	Questions	Tolokers
GCN_BASE	73.36 ± 0.73	50.30 ± 0.39	89.64 ± 0.53	76.33 ± 0.93	84.10 ± 0.99
GCN_ENS	75.25 ± 0.73	50.68 ± 0.47	89.90 ± 0.55	76.73 ± 1.08	84.10 ± 0.81
GCN_GNNM	77.74 ± 0.54	49.98 ± 0.57	88.77 ± 0.62	77.50 ± 1.02	83.31 ± 0.65
SAGE_BASE	86.68 ± 0.55	53.78 ± 0.61	93.72 ± 0.54	73.66 ± 0.86	82.97 ± 0.41
SAGE_ENS	87.47 ± 0.47	54.93 ± 0.44	93.87 ± 0.44	74.39 ± 0.97	83.04 ± 0.40
SAGE_GNNM	90.04 ± 0.46	55.41 ± 0.45	**94.01 ± 0.47**	77.02 ± 1.12	83.47 ± 0.46
GAT_BASE	82.78 ± 0.87	51.71 ± 0.51	92.36 ± 0.49	77.17 ± 0.67	84.02 ± 0.58
GAT_ENS	83.89 ± 0.40	53.02 ± 0.62	92.77 ± 0.58	78.00 ± 0.91	84.53 ± 0.58
GAT_GNNM	82.71 ± 0.48	52.89 ± 0.44	91.11 ± 0.74	**79.13 ± 0.89**	83.94 ± 0.71
GAT-sep_BASE	89.23 ± 0.58	54.43 ± 0.47	93.39 ± 0.48	77.57 ± 0.63	84.04 ± 0.53
GAT-sep_ENS	90.14 ± 0.42	55.58 ± 0.51	93.71 ± 0.41	78.04 ± 0.84	84.35 ± 0.54
GAT-sep_GNNM	**91.19 ± 0.33**	**55.77 ± 0.44**	93.70 ± 0.46	79.10 ± 0.92	**84.83 ± 0.65**
GT_BASE	88.25 ± 0.53	53.82 ± 0.44	92.30 ± 0.52	77.78 ± 0.75	82.92 ± 0.70
GT_ENS	89.01 ± 0.43	55.11 ± 0.39	92.59 ± 0.55	78.18 ± 0.86	83.80 ± 0.59
GT_GNNM	90.25 ± 0.31	55.53 ± 0.43	92.83 ± 0.44	78.66 ± 0.94	84.19 ± 0.63
GT-sep_BASE	87.46 ± 0.48	53.75 ± 0.29	92.00 ± 0.60	77.27 ± 0.72	83.02 ± 0.81
GT-sep_ENS	88.21 ± 0.37	54.90 ± 0.44	92.48 ± 0.50	77.83 ± 0.85	83.79 ± 0.86
GT-sep_GNNM	89.81 ± 0.44	55.32 ± 0.46	92.50 ± 0.28	78.62 ± 0.74	84.42 ± 0.79
ResNet_BASE	65.88 ± 0.34	48.21 ± 0.45	50.69 ± 1.19	70.52 ± 0.67	72.91 ± 1.12
ResNet_ENS	66.16 ± 0.49	49.52 ± 0.61	50.95 ± 1.08	70.83 ± 0.78	73.25 ± 1.03
ResNet_GNNM	66.01 ± 0.45	49.44 ± 0.44	50.67 ± 1.18	71.52 ± 0.86	73.80 ± 1.04

Table 1: Test performance on the five heterophilous node classification benchmarks. We report mean ± standard deviation over the 10 official splits (%). The metric is accuracy for Roman-empire and Amazon-ratings, and ROC-AUC for Minesweeper, Questions, and Tolokers. For each dataset, the best result is shown in **bold** and the second best is underlined.

5.1 Main comparison

For both explicit ensembles and GNNM, we use $M = 4$ prediction instances in the main comparison. Thus, each ENS result averages four independently trained GNNs, while each GNNM result uses four BatchEnsemble instances.

Table 1 compares vanilla models, explicit ensembles, and GNNM variants across all datasets and backbone architectures. The best result on every dataset is achieved by a GNNM variant. The strength of the effect, however, varies by backbone (Table 2): GNNM beats both the vanilla model and the explicit ensemble on all five datasets for SAGE, GT, and GT-sep and on four of five for GAT-sep. In particular, GAT-sep_GNNM achieves the best score on Roman-empire, Amazon-ratings, and Tolokers, SAGE_GNNM achieves the best score on Minesweeper and GAT_GNNM achieves the best score on Questions. Thus, the strongest method on each benchmark is not a single model or an explicit ensemble, but a projector-level BatchEnsemble variant. The improvement is not uniform, however: GNNM is worse than BASE in 7 of the 35 backbone–dataset pairs and worse than ENS in 11 of 35, mostly for GCN, GAT, and ResNet.

To separate dataset-level wins from backbone-level behavior, we additionally aggregate performance across the five datasets for each backbone family. Table 2 reports the average score of BASE, explicit ensemble, and GNNM variants for each backbone. The GNNM variant improves over BASE on average for every backbone family and outperforms the explicit ensemble in six out of the seven cases, with GAT being the only exception. Averaged over all backbone-dataset pairs, GNNM achieves the highest mean score. This supports the central claim that projector-level BatchEnsemble is a parameter-efficient alternative to ordinary ensembling that, on average, is at least as accurate.

Statistical significance. We tested significance of the improvements at the level of the $7 \times 5 = 35$ backbone–dataset cells: each cell is reduced to its mean over the 10 splits, so the correlated splits within a cell are not counted as independent observations. We run a one-sided Wilcoxon signed-rank test on the per-cell standardized differences (each cell's paired difference divided by its across-split

Backbone	BASE	ENS	GNNM	GNNM wins
GCN	74.75	75.33	**75.46**	2/5
SAGE	78.16	78.74	**79.99**	5/5
GAT	77.61	**78.44**	77.96	1/5
GAT-sep	79.73	80.36	**80.92**	4/5
GT	79.01	79.74	**80.29**	5/5
GT-sep	78.70	79.44	**80.13**	5/5
ResNet	61.64	62.14	**62.29**	2/5
All	75.66	76.31	**76.72**	24/35

Table 2: Average performance across the five benchmarks for each backbone family. Each value is the arithmetic mean of the dataset scores from Table 1 (accuracy and ROC-AUC are averaged together, so the averages are descriptive only). "GNNM wins" is the number of datasets on which GNNM has a higher mean score than both BASE and ENS of the same backbone. The best result in each row is shown in **bold**.

Test (unit)	vs. BASE	vs. ENS
Wilcoxon, std. (35 cells)	4.8×10^{-6}	8.3×10^{-3}
Friedman+Holm (35 cells)	8.2×10^{-6}	2.1×10^{-2}
Sign-flip, backbone clusters (7)	0.008	0.039
Sign-flip, dataset clusters (5)	0.063	0.16
Two-way bootstrap, 95% CI	$[0.27, 1.86]$	$[-0.25, 1.10]$

Table 3: Significance of the improvements of GNNM, from the least to the most conservative treatment of the dependence between backbone–dataset cells. Mean paired differences are $+1.06$ points over BASE and $+0.41$ points over ENS.

standard deviation). Under this test GNNM improves significantly over the vanilla BASE backbone ($p = 4.8 \times 10^{-6}$) and over ENS ($p = 8.3 \times 10^{-3}$).

We further corroborate this with the standard protocol for comparing methods across many tasks (Demšar, 2006): a Friedman omnibus test followed by Holm-corrected post-hoc two-sided Wilcoxon signed-rank tests over the three pairwise comparisons. This test is purely rank-based, so it is invariant to the accuracy/ROC-AUC scale mismatch and requires no standardization. The omnibus rejects equal performance ($\chi^2_F = 31.94$, $p = 1.2 \times 10^{-7}$). The mean ranks are 1.514/1.714/2.771 for GNNM/ENS/BASE (lower is better, as shown in Appendix B for the computation). The Holm-corrected post-hoc tests confirm that GNNM significantly outranks both BASE ($p = 8.2 \times 10^{-6}$) and ENS ($p = 2.1 \times 10^{-2}$).

Both tests treat the 35 cells as independent, although cells of the same dataset share the graph and splits and cells of the same backbone share the architecture. We therefore also run exact sign-flip tests on cluster means and a two-way cluster bootstrap over backbones and datasets (Table 3, Appendix B). The gain over BASE is positive for all seven backbones ($p = 0.008$) and its bootstrap interval excludes zero, but with only five datasets it is not significant at the dataset level ($p = 0.063$, whereas the minimum attainable p is 0.031). The gain over ENS is positive for six of seven backbones ($p = 0.039$) but not significant at the dataset level ($p = 0.16$). We therefore conclude that GNNM clearly improves over the vanilla backbone and performs on par with or better than the explicit ensemble, rather than being uniformly stronger than it.

5.2 Number of BatchEnsemble instances

We next study the effect of the number of BatchEnsemble instances. This ablation evaluates whether the gains of GNNM are driven only by increasing the number of predictions, or whether the architecture remains effective across different ensemble sizes. We compare $M \in \{2, 4, 8\}$ while keeping the backbone architecture and all other hyperparameters fixed.

For this ablation, we use Roman-empire with $L = 5$ message-passing layers and report test accuracy over the first 5 splits. The simplified setup was used to speed up the ablation experiments. The results are shown in Table 4. Increasing the number of BatchEnsemble instances consistently improves performance for all three backbones. The best result is always obtained with $M = 8$, while $M = 4$ already provides a clear improvement over $M = 2$. This suggests that additional BatchEnsemble instances are useful across different GNN backbones, although the improvement becomes less pronounced as M grows.

Backbone	$M = 2$	$M = 4$	$M = 8$
SAGE	87.71 ± 0.56	89.50 ± 0.33	$\mathbf{90.83 \pm 0.36}$
GAT-sep	89.91 ± 0.29	90.85 ± 0.39	$\mathbf{91.37 \pm 0.48}$
GT-sep	87.70 ± 0.41	89.63 ± 0.25	$\mathbf{90.84 \pm 0.31}$

Table 4: Effect of the number of BatchEnsemble instances on Roman-empire. We report test accuracy averaged over 5 splits. The best result for each backbone is shown in **bold**.

Backbone	Ours	Xavier-all
SAGE	$\mathbf{89.50 \pm 0.33}$	87.20 ± 0.27
GAT-sep	$\mathbf{90.98 \pm 0.18}^*$	89.58 ± 0.53
GT-sep	$\mathbf{89.63 \pm 0.25}$	88.04 ± 0.79

Table 5: Effect of BatchEnsemble initialization on Roman-empire with $M = 4$ (5 splits). *Separate run of the same configuration as Table 4, as discussed in the text.

These results support the use of $M = 4$ in the main comparison as a practical setting — it captures a substantial part of the gain from increasing the number of prediction instances, while avoiding the larger cost of $M = 8$.

5.3 Initialization strategy

We also study the initialization of the BatchEnsemble factors. As described in Section 2.4, our default initialization follows the TabM-style scheme: most BatchEnsemble projectors start from an identity-like parameterization, while the first projector introduces instance-specific variation. This design keeps the initial GNN pipeline close to the corresponding vanilla model while still allowing different prediction instances to emerge.

We compare this default initialization with the default initialization used for the other linear blocks of the network, which we call Xavier-all: the modulation factors r_m of both BatchEnsemble projectors are initialized with Xavier-uniform initialization, like the shared weight matrices, while $s_m = 1$ and $b_m = 0$ as before. Note that Xavier-uniform factors are small and of random sign, so this scheme changes both the diversity between instances and the scale of the initial activations. This ablation evaluates whether the TabM-style asymmetric initialization is important for transferring parameter-efficient ensembling from tabular MLPs to GNN pipelines.

For this experiment, we fix $M = 4$ and use the same Roman-empire setup as in the previous ablation. The results are reported in Table 5. The default TabM-style initialization is consistently better for all three backbones. The "Ours" column is a separate run of the $M = 4$ configuration of Table 4 with the same seeds, and SAGE and GT-sep reproduce exactly, while GAT-sep differs slightly (90.98 vs. 90.85) due to non-deterministic GPU scatter operations in attention.

This shows that the asymmetric TabM-style initialization is not a minor implementation detail. The first projector must introduce sufficient instance-specific variation at the beginning of training, while the output projector should allow the model to decide which dimensions should be ensembled and which should not. Since Xavier-all changes both diversity and scale, the ablation does not separate these two effects.

5.4 Cost analysis

Finally, we compare the computational cost of BASE, explicit ensembles, and GNNM. We measure the number of trainable parameters, mean training step time, and peak GPU memory. For each dataset and variant, the values in Table 6 are averaged over 8 backbones (the seven backbones above and TAG) and 5 depths, giving 40 configurations per variant.

The results show that GNNM has almost the same number of parameters as BASE, while $\text{ENS}_{M=4}$ requires four full model copies. The training step time of GNNM scales approximately linearly with the number of BatchEnsemble instances (Appendix C), and $\text{GNNM}_{M=4}$ is close to $\text{ENS}_{M=4}$ in wall-clock time (0.3–2.5% slower). This is expected because both methods compute four prediction paths, so GNNM is not cheaper than an explicit ensemble in computation. Peak memory of GNNM is almost constant in M (Appendix C) because the instances are processed one after another with gradient accumulation (Section 3) and the additional BatchEnsemble parameters are very small. The main difference is therefore not step time, but parameter efficiency. $\text{GNNM}_{M=4}$ uses approximately four times fewer parameters than $\text{ENS}_{M=4}$, has similar peak memory (lower on four of the five datasets), and is on average at least as accurate in the main results.

	Params, M			Step time, ms			Peak memory, MB		
Dataset	BASE	ENS	GNNM	BASE	ENS	GNNM	BASE	ENS	GNNM
Roman-empire	4.01	16.04	4.02	51.3	204.4	209.6	2044	2228	2164
Amazon-ratings	4.00	16.01	4.01	69.8	278.5	284.0	3522	3706	3636
Minesweeper	3.85	15.40	3.86	32.0	126.8	128.5	1501	1679	1543
Tolokers	3.85	15.41	3.86	164.8	658.7	660.6	12776	12953	12818
Questions	4.00	16.00	4.01	135.7	542.1	552.8	6201	6385	6420

Table 6: Cost analysis for BASE, ENS$_{M=4}$, and GNNM$_{M=4}$. Parameters are reported in millions, step time in milliseconds, and peak GPU memory in MB. The full table with GNNM$_{M=2}$ and GNNM$_{M=8}$ is given in Appendix C.

6 Conclusion

We introduced GNNM, a projector-level BatchEnsemble architecture for graph neural networks. The method keeps the message-passing backbone shared and applies BatchEnsemble only to the input and output projectors, allowing a single GNN pipeline to produce multiple predictions without duplicating the full graph backbone. This design transfers the TabM-style parameter-efficient ensembling principle from tabular models to graph learning.

Our experiments show that GNNM is a parameter-efficient alternative to explicit ensembling. On the five node classification benchmarks considered in this work, the best result on every dataset is achieved by a GNNM variant. Averaged across backbone families, GNNM improves over the corresponding vanilla models and usually outperforms explicit ensembles built from the same GNN backbone, although the advantage over explicit ensembles is not statistically significant once the dependence between results on the same dataset is taken into account. These results indicate that projector-level BatchEnsemble can match ordinary model stacking at a fraction of its parameter count.

The ablation studies further show that the number of BatchEnsemble instances and the initialization strategy are important design choices. In particular, the TabM-style initialization provides a stable way to introduce instance-specific variation while keeping the initial model close to the underlying vanilla GNN. The cost analysis confirms the practical motivation of the method: GNNM avoids full duplication of the message-passing stack and therefore needs about four times fewer parameters than an explicit ensemble, while its computation remains comparable to that of the explicit ensemble.

7 Limitations

This work focuses on node classification benchmarks. Although the proposed modification is not specific to node classification, its behavior on other graph learning tasks remains to be evaluated.

Another limitation is that we study GNNM mainly as an alternative to ordinary explicit ensembles. A further direction is to combine both ideas by building a simple explicit ensemble of several independently trained GNNM models. Such a model would stack two levels of ensembling: BatchEnsemble instances inside each GNNM and ordinary averaging across multiple GNNM models. This could further improve predictive quality, although at a higher computational and storage cost.

Reproducibility statement

All hyperparameters, the model-selection protocol, and the random seeds are listed in Appendix A. The datasets and splits are the public ones of Platonov et al. (2023). The hardware and the timing procedure are described in Appendix E, and the statistical tests in Appendix B. The code for all variants, ablations, and significance tests will be released upon publication.

REFERENCES

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Gangda Deng, Yuxin Yang, Ömer Faruk Akgül, Hanqing Zeng, Yinglong Xia, Rajgopal Kannan, and Viktor Prasanna. Training diverse graph experts for ensembles: A systematic empirical study. *arXiv preprint arXiv:2510.18370*, 2025.

Michael W. Dusenberry, Ghassen Jerfel, Yeming Wen, Yi-An Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable Bayesian neural nets with rank-1 factors. In *International Conference on Machine Learning (ICML)*, 2020.

Dominik Fuchsgruber, Tom Wollschläger, Johannes Bordne, and Stephan Günnemann. Uncertainty estimation for heterophilic graphs through the lens of information theory. In *International Conference on Machine Learning (ICML)*, 2025.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. TabM: Advancing tabular deep learning with parameter-efficient ensembling. In *International Conference on Learning Representations (ICLR)*, 2025.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M. Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *International Conference on Learning Representations (ICLR)*, 2021.

Hans Hao-Hsun Hsu, Yuesong Shen, Christian Tomani, and Daniel Cremers. What makes graph neural networks miscalibrated? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get M for free. *International Conference on Learning Representations (ICLR)*, 2017.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Olivier Laurent, Adrien Lafage, Enzo Tartaglione, Geoffrey Daniel, Jean-Marc Martinez, Andrei Bursuc, and Gianni Franchi. Packed-ensembles for efficient uncertainty estimation. In *International Conference on Learning Representations (ICLR)*, 2023.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *ICML Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020.

Amrit Nagarajan, Jacob R. Stevens, and Anand Raghunathan. Efficient ensembles of graph neural networks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022.

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *International Conference on Learning Representations (ICLR)*, 2023.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. In *Relational Representation Learning Workshop, NeurIPS*, 2018.

Maximilian Stadler, Bertrand Charpentier, Simon Geisler, Daniel Zügner, and Stephan Günnemann. Graph posterior network: Bayesian predictive uncertainty for node classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Tirtha Vinchurkar, Kareem Abdelmaqsoud, and John R. Kitchin. Uncertainty quantification in graph neural networks with shallow ensembles. *arXiv preprint arXiv:2504.12627*, 2025.

Xiao Wang, Hongrui Liu, Chuan Shi, and Cheng Yang. Be confident! towards trustworthy graph neural networks via confidence calibration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Yeming Wen, Dustin Tran, and Jimmy Ba. BatchEnsemble: An alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations (ICLR)*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Xin Zhang, Daochen Zha, and Qiaoyu Tan. E2GNN: Efficient graph neural network ensembles for semi-supervised classification. *arXiv preprint arXiv:2405.03401*, 2024.

A Hyperparameter Details

For the main experiments, we keep the setup identical across vanilla GNNs, explicit ensembles, and GNNM variants. On the first split, we grid search over the number of layers and select the best value on validation, after which the selected value is reused for the remaining splits and the test score is taken from the checkpoint with the best validation score. All other hyperparameters are fixed across all compared variants. Therefore, the comparison changes only the ensembling mechanism. Hyperparameters were taken the same as in the benchmark paper (Table 7). For split i, the random seed of BASE and GNNM is i, and the four members of ENS use seeds $1000i, \ldots, 1000i + 3$.

To reduce the computational cost of the ablation studies, we use Roman-empire only, select three substantially different backbones (SAGE, GAT-sep, and GT-sep), fix $L = 5$, and report results over the first 5 splits.

Parameter	Value
Number of splits	10
Layer search	$\{1, 2, 3, 4, 5\}$
Hidden dimension	512
Learning rate	$3 \cdot 10^{-5}$
Optimizer	AdamW
Weight decay	0
Dropout	0.2
Normalization	LayerNorm
Activation	GELU
Attention heads	8
Maximum training steps	5000
Early stopping patience	300 steps
Explicit ensemble size	$M = 4$ models
GNNM instances	$M = 4$

Table 7: Hyperparameter setup for the main experiments.

B STATISTICAL TESTS

All tests use the per-split results of Table 1. Each backbone–dataset cell is first reduced to its mean over the 10 splits.

Wilcoxon and Friedman tests. The standardized Wilcoxon test divides each cell's mean paired difference by the across-split standard deviation of that difference and applies a one-sided exact Wilcoxon signed-rank test to the 35 values. The Friedman test ranks BASE, ENS, and GNNM within each cell. With $N = 35$ cells and $k = 3$ variants, $\chi^2_F = \frac{12N}{k(k+1)}\big[\sum_j \bar{R}_j^2 - \frac{k(k+1)^2}{4}\big] = 31.94$ for the mean ranks $\bar{R} = (1.514, 1.714, 2.771)$. The ranks are computed from unrounded cell means, so they can differ slightly from ranks computed from the rounded values in Table 1. Post-hoc tests are two-sided exact Wilcoxon signed-rank tests on the cell means with Holm's step-down correction over the three pairs (GNNM–BASE, GNNM–ENS, ENS–BASE).

Cluster-level sign-flip tests. For a clustering by dataset or by backbone, we average the paired differences within each cluster and compute the exact one-sided p-value of their sum over all 2^C sign flips. The per-dataset mean differences (Roman-empire, Amazon-ratings, Minesweeper, Questions, Tolokers) are $2.02, 1.19, -0.07, 1.61, 0.57$ for GNNM$-$BASE and $1.09, 0.09, -0.38, 1.08, 0.16$ for GNNM$-$ENS. The per-backbone mean differences (GCN, SAGE, GAT, GAT-sep, GT, GT-sep, ResNet) are $0.71, 1.83, 0.35, 1.19, 1.28, 1.44, 0.65$ and $0.13, 1.25, -0.49, 0.55, 0.56, 0.69, 0.15$.

Two-way cluster bootstrap. In each of 20,000 replicates, we resample the 7 backbones and the 5 datasets independently with replacement and record the mean paired difference over the resulting sub-grid, and we report percentile intervals. The sign-flip and bootstrap analyses average accuracy and ROC-AUC differences in points, so, unlike the rank-based tests, they are not invariant to the difference between the two metrics.

C FULL COST ANALYSIS

D GRAPH-CLASSIFICATION BENCHMARK

To check that the GNNM recipe is not specific to node classification, we run the same three-way comparison (BASE, ENS, GNNM) on a graph-level task. We use the PROTEINS dataset from the TUDataset collection (Morris et al., 2020). The backbone is GraphSAGE (Hamilton et al., 2017) with a global mean-pooling readout. All three variants share an identical hyperparameter grid, so the comparison isolates the ensembling mechanism.

As we can see in Table 9, GNNM obtains the highest mean accuracy, but the differences between the three variants are smaller than the fold-to-fold standard deviation, so this experiment shows that GNNM does not hurt on this task rather than that it improves it.

E COMPUTATIONAL RESOURCES

The main experiments were conducted on a server with two NVIDIA H200 GPUs. The full set of main experiments took approximately five days to complete. The ablation studies were conducted on a single NVIDIA H200 GPU and required approximately 12 hours per run.

For the cost analysis, each configuration is trained on the first split for 50 warm-up steps followed by 200 measured steps with full-graph training. Step time is the mean wall-clock time of one training step (forward pass, backward pass, and optimizer update, with one step of each of the four members for ENS), measured with CUDA synchronization before and after the step. Peak memory is the maximum GPU memory allocated by PyTorch during these steps.

Dataset	Variant	Params, M	Step time, ms	Peak memory, MB
Amazon-ratings	BASE	4.00	69.8	3522
Amazon-ratings	ENS$_{M=4}$	16.01	278.5	3706
Amazon-ratings	GNNM$_{M=2}$	4.01	142.2	3636
Amazon-ratings	GNNM$_{M=4}$	4.01	284.0	3636
Amazon-ratings	GNNM$_{M=8}$	4.02	567.5	3636
Minesweeper	BASE	3.85	32.0	1501
Minesweeper	ENS$_{M=4}$	15.40	126.8	1679
Minesweeper	GNNM$_{M=2}$	3.85	64.5	1543
Minesweeper	GNNM$_{M=4}$	3.86	128.5	1543
Minesweeper	GNNM$_{M=8}$	3.86	256.6	1543
Questions	BASE	4.00	135.7	6201
Questions	ENS$_{M=4}$	16.00	542.1	6385
Questions	GNNM$_{M=2}$	4.00	276.6	6420
Questions	GNNM$_{M=4}$	4.01	552.8	6420
Questions	GNNM$_{M=8}$	4.01	1105.5	6420
Roman-empire	BASE	4.01	51.3	2044
Roman-empire	ENS$_{M=4}$	16.04	204.4	2228
Roman-empire	GNNM$_{M=2}$	4.01	105.0	2164
Roman-empire	GNNM$_{M=4}$	4.02	209.6	2164
Roman-empire	GNNM$_{M=8}$	4.02	418.8	2164
Tolokers	BASE	3.85	164.8	12776
Tolokers	ENS$_{M=4}$	15.41	658.7	12953
Tolokers	GNNM$_{M=2}$	3.85	330.6	12818
Tolokers	GNNM$_{M=4}$	3.86	660.6	12818
Tolokers	GNNM$_{M=8}$	3.86	1321.0	12818

Table 8: Cost analysis. Parameters are reported in millions, step time in milliseconds, and peak GPU memory in MB. Step time includes the forward pass, backward pass, and optimizer update (for ENS, of all four members), and the measurement procedure is given in Appendix E.

Variant	Test accuracy (%)	Params
BASE	73.35 ± 2.62	$1\times$
ENS (M=4)	72.90 ± 4.05	$4\times$
GNNM (M=4)	$\mathbf{73.62 \pm 3.31}$	$\sim 1.06\times$

Table 9: Graph classification on PROTEINS (TUDataset). We report the mean test accuracy $\pm$ standard deviation over the 10 outer folds.

F AI USAGE

Generative AI tools were used only for text polishing, including minor improvements to grammar, phrasing, and readability. All scientific content, experiments, analyses, and conclusions were produced by the authors.