BLISS: Bandit Layer Importance Sampling Strategy for Efficient Training of Graph Neural Networks

Omar Alsaqa
Wilfrid Laurier University
Waterloo, Ontario, Canada
o.alsaqa@gmail.com

Thi Linh Hoang
Singapore Management University
Singapore
tlhoang@smu.edu.sg

Muhammed Fatih Balin
Georgia Institute of Technology
Atlanta, GA, USA
balin@gatech.edu

Abstract

Graph Neural Networks (GNNs) are powerful tools for learning from graph-structured data, but their application to large graphs is hindered by computational costs. The need to process every neighbor for each node creates memory and computational bottlenecks. To address this, we introduce BLISS, a Bandit Layer Importance Sampling Strategy. It uses multi-armed bandits to dynamically select the most informative nodes at each layer, balancing exploration and exploitation to ensure comprehensive graph coverage. Unlike existing static sampling methods, BLISS adapts to evolving node importance, leading to more informed node selection and improved performance. It demonstrates versatility by integrating with both Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs), adapting its selection policy to their specific aggregation mechanisms. Experiments show that BLISS maintains or exceeds the accuracy of full-batch training.

1 Introduction

Graph Neural Networks (GNNs) are powerful tools for learning from graph-structured data, enabling applications such as personalized recommendations Ying et al. [2018], Wang et al. [2019], drug discovery Lim et al. [2019], Merchant et al. [2023], image understanding Han et al. [2022, 2023], and enhancing Large Language Models (LLMs) Yoon et al. [2023], Tang et al. [2023], Chen et al. [2023]. Architectures like GCNs and GATs have addressed early limitations in capturing long-range dependencies.

However, training GNNs on large graphs remains challenging due to prohibitive memory and computational demands, primarily because considering all neighbor nodes for each node leads to excessive memory and computational costs. While mini-batching, common in deep neural networks, can mitigate memory issues, uninformative mini-batches can lead to: 1) **Sparse representations**: Nodes may be isolated, neglecting crucial connections and resulting in poor representations. 2) **Neighborhood explosion**: A node's receptive field grows exponentially with layers, making recursive neighbor aggregation computationally prohibitive even for single-node mini-batches.

Efficient neighbor sampling is essential to address these challenge. Techniques include random selection, feature- or importance-based sampling, and adaptive strategies learned during training. They fall into three categories: **(1) Node-wise sampling**, which selects neighbors per node to reduce cost but risks redundancy (e.g., GraphSAGE Hamilton et al. [2017], VR-GCN Chen et al. [2017], BS-GNN Liu et al. [2020]); **(2) Layer-wise sampling**, which samples neighbors jointly at each layer for efficiency and broader coverage but may introduce bias (e.g., FastGCN Chen et al. [2018], LADIES Zou et al. [2019], LABOR Balin and Çatalyürek [2023]); and **(3) Sub-graph sampling**, which uses induced subgraphs for message passing, improving efficiency but potentially losing global context if reused across layers (e.g., Cluster-GCN Chiang et al. [2019], GraphSAINT Zeng et al. [2019]).

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

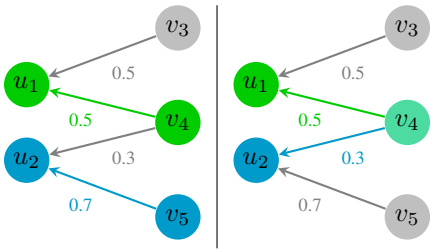

Figure 1: The figures illustrate Node-wise vs. Layer-wise sampling. **Left:** Node-wise sampling selects nodes per target node, often causing redundancy (e.g., v_4 sampled for both u_1 and u_2), higher sampling rates (e.g., v_4, v_5), and missing edges (e.g., u_2–v_4). **Right:** Layer-wise sampling considers all nodes in the previous layer, preserving structure and connectivity while sampling fewer nodes.

Our key contributions are: (1) **Modeling neighbor selection as a layer-wise bandit problem:** Each edge represents an "arm" and the reward is based on the neighbor's contribution to reducing the variance of the representation estimator. (2) **Applicability to Different GNN Architectures:** BLISS is designed to be compatible with various GNN architectures, including GCNs and GATs.

The remainder is organized as follows: section 2 describes BLISS; section 3 reports results; section 4 concludes. A detailed background and related work appear in appendices B and C.

2 Proposed Method

2.1 Bandit-Based Layer Importance Sampling Strategy (BLISS)

BLISS selects informative neighbors per node and layer via a policy-based approach, guided by a dynamically updated sampling distribution driven by rewards reflecting each neighbor's contribution to node representation. Using bandit algorithms, BLISS balances exploration and exploitation, adapts to evolving embeddings, and maintains scalability on large graphs. Traditional node sampling often fails to manage this trade-off or adapt to changing node importance, reducing accuracy and scalability. While Liu et al. [2020] framed node-wise sampling as a bandit problem, BLISS extends it to layer-wise sampling, leveraging inter-layer information flow and reducing redundancy (see fig. 1).

Initially, edge weights $w_{ij} = 1$ for all $j \in \mathcal{N}_i$, with sampling probabilities q_{ij} set proportionally. BLISS proceeds top-down from the final layer L, computing layer-wise sampling probabilities p_j for nodes in layer l. These are passed to algorithm 4, which selects k nodes. The GNN then performs a forward pass, where each node i aggregates from sampled neighbors j_s to approximate its representation $\hat{\mu}_i$:

$$p_j = \sqrt{\sum_i \left(\frac{q_{ij}}{\sum_{k \in \mathcal{N}_i} q_{ik}} \right)^2} \qquad (1)$$

$$h_i = \frac{1}{k} \sum_{s=1}^{k} \frac{\alpha_{ijs}}{q_{ijs}} \hat{h}_{j_s} \qquad (2)$$

Here, $j_s \sim q_i$ denotes the s-th sampled neighbor of node i, drawn from the per-node sampling distribution q_i. This process updates node representations h_j. The informativeness of neighbors is quantified as a reward r_{ij}, and the estimated rewards $\hat{r}_{ij}$ are calculated as:

$$r_{ij} = \frac{\alpha_{ij}^2}{k \cdot q_j^2} \|h_j\|_2^2, \qquad (3)$$

$$\hat{r}_{ij}^{(t)} = \frac{r_{ij}^{(t)}}{q_j^{(t)}} \quad \text{if } j \in S_i^t \qquad (4)$$

where S_i^t is the set of sampled neighbors at step t, α_{ij} is the aggregation coefficient, and h_j is the node embedding. The edge weights w_{ij} and sampling probabilities q_{ij} are updated using the EXP3 algorithm (see algorithm 5). The edge weights are updated as follows:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} \exp\left(\frac{\delta \hat{r}_{ij}^{(t)}}{|\mathcal{N}_i|} \right) \qquad (5)$$

$$q_{ij}^{(t+1)} = (1 - \eta) \frac{w_{ij}^{(t+1)}}{\sum_{j \in \mathcal{N}_i} w_{ij}^{(t+1)}} + \frac{\eta}{|\mathcal{N}_i|} \qquad (6)$$

where δ is a scaling factor and η is the bandit learning rate.

BLISS operates through an iterative process of four steps: (1) dynamically selecting nodes at each layer via a bandit algorithm (e.g., EXP3) that assigns sampling probabilities, (2) estimating node representations by aggregating from sampled neighbors using Monte Carlo estimation, (3) performing standard GNN message passing with these samples, and (4) calculating rewards based on neighbor contributions to update the bandit policy and refine future sampling distributions. For the detailed algorithm check algorithm 2.

2.2 Adapting to Graph Attention

BLISS: We extend BLISS to attentive GNNs, following Liu et al. [2020]. With only a sampled neighbor set S_i, true normalized attention α_{ij} is unavailable. We compute unnormalized scores $\tilde{\alpha}_{ij}$ and define adjusted *feedback* attention: $\alpha'_{ij} = \sum_{j \in S_i} q_{ij} \frac{\tilde{\alpha}_{ij}}{\sum_{j \in S_i} \tilde{\alpha}_{ij}}$ where q_{ij} is the bandit-determined sampling probability of edge e_{ij}. We use $\sum_{j \in S_i} q_{ij}$ as a surrogate for the normalization over the full neighborhood N_i, thus approximating α_{ij} while properly weighting sampled neighbors within the attention mechanism.

PLADIES:Applying LADIES to attentive GNNs (e.g., GATs) requires preserving at least one neighbor per node after sampling to respect attention's dependence on neighbor information. The PLADIES edge-sampling procedure, adapted from Balin and Çatalyürek [2023] and detailed in algorithm 4, first computes initial probabilities (p_j), then iteratively adjusts a scaling factor (c) so the sum of clipped probabilities approaches the target sample size (k). Probabilities for seed nodes V_{skip} are set to ∞, guaranteeing selection and creating "skip connections," ensuring each node retains a neighbor for attention while enabling LADIES to leverage attention efficiently.

3 Experiments

3.1 Datasets

We evaluate the performance of each method in the node prediction task on the following datasets: Cora, Citeseer Sen et al. [2008], Pubmed Namata et al. [2012], Flickr, Yelp Zeng et al. [2019], and Reddit Hamilton et al. [2017]. More details of the benchmark datasets are given in table 3.

3.2 Experiment Settings

We compare BLISS with PLADIES, a strong baseline among existing layer-wise sampling algorithms. The code for both BLISS and PLADIES is publicly available.[1]

Model and Training. We use 3-layer GNNs (GraphSAGE and GATv2) with a hidden dimension of 256. Models are trained with the ADAM optimizer with a learning rate of 0.002. For bandit experiments, we set $\eta = 0.4$ and $\delta = \eta/10^6$ to prevent large updates.

Sampling Parameters. Batch sizes and fanouts for each dataset are listed in table 4. For smaller datasets (Citeseer, Cora, Pubmed), a small batch size is chosen to ensure the sampler does not process all training nodes in a single step (training nodes: 120, 140, and 60 respectively). For larger datasets (Flickr, Yelp, Reddit), relatively small batch sizes are used to accommodate limited computational resources (tested on a P100 GPU with 16GB VRAM). An incremental fanout configuration ensures sufficient local neighborhood aggregation: the first layer's fanout is set to four times the batch size, and subsequent layers' fanouts are twice the preceding layer's.

Evaluation. For all methods and datasets, training is conducted 5 times with different seeds, and the mean and standard deviation of the F1-score on the test set are reported. The number of training steps for each dataset is specified in table 4. We run the experiments on GraphSAGE Hamilton et al. [2017] and GATv2 Brody et al. [2021].

[1]The code implementation is available at: `https://github.com/linhthi/BLISS-GNN`

Table 1: Comparison of F1-scores (mean ± standard deviation) for BLISS and PLADIES samplers on six datasets using Graph Attention Networks (GAT) and GraphSAGE (SAGE) architectures.

Dataset	Sampler	Train		Validation		Test	
		GAT	SAGE	GAT	SAGE	GAT	SAGE
citeseer	BLISS	0.927 ± 0.005	0.947 ± 0.013	0.712 ± 0.004	0.598 ± 0.028	**0.706 ± 0.002**	0.580 ± 0.032
	PLADIES	0.912 ± 0.007	0.963 ± 0.016	0.699 ± 0.008	0.616 ± 0.020	**0.683 ± 0.005**	0.601 ± 0.017
cora	BLISS	0.989 ± 0.002	0.983 ± 0.005	0.802 ± 0.005	0.785 ± 0.005	**0.813 ± 0.004**	0.795 ± 0.009
	PLADIES	0.989 ± 0.003	0.981 ± 0.005	0.800 ± 0.004	0.767 ± 0.011	**0.809 ± 0.003**	0.772 ± 0.014
flickr	BLISS	0.515 ± 0.003	0.516 ± 0.002	0.511 ± 0.003	0.503 ± 0.001	**0.511 ± 0.002**	0.503 ± 0.002
	PLADIES	0.511 ± 0.006	0.515 ± 0.001	0.507 ± 0.005	0.504 ± 0.001	**0.507 ± 0.005**	0.505 ± 0.001
pubmed	BLISS	0.907 ± 0.008	0.807 ± 0.063	0.748 ± 0.006	0.594 ± 0.047	**0.731 ± 0.007**	0.597 ± 0.057
	PLADIES	0.910 ± 0.008	0.760 ± 0.042	0.750 ± 0.014	0.571 ± 0.038	**0.718 ± 0.013**	0.557 ± 0.042
reddit	BLISS	0.953 ± 0.001	0.979 ± 0.001	0.949 ± 0.001	0.962 ± 0.000	0.949 ± 0.001	**0.962 ± 0.000**
	PLADIES	0.954 ± 0.002	0.979 ± 0.001	0.951 ± 0.001	0.962 ± 0.000	0.950 ± 0.001	**0.962 ± 0.000**
yelp	BLISS	0.540 ± 0.002	0.530 ± 0.005	0.538 ± 0.002	0.527 ± 0.005	**0.540 ± 0.002**	0.529 ± 0.005
	PLADIES	0.540 ± 0.002	0.503 ± 0.009	0.537 ± 0.002	0.501 ± 0.009	**0.539 ± 0.002**	0.502 ± 0.009

Baseline Justification. We compare BLISS against PLADIES from Balin and Çatalyürek [2023] because it represents the state-of-the-art in layer-wise sampling, which is the specific category BLISS belongs to. While other sampling methods like GraphSAINT Zeng et al. [2019] (subgraph sampling) or GCN-BS Liu et al. [2020] (node-wise bandit sampling) exist, direct comparison would require different experimental setups or fall outside the scope of layer-wise sampling. Our goal is to achieve accuracy comparable to full-batch training while maintaining scalability, which PLADIES also aims for within the layer-wise paradigm.

3.3 Results

Our experiments confirm that BLISS, a dynamic layer-wise sampling strategy, consistently outperforms the PLADIES sampler across multiple benchmark datasets and GNN architectures (GAT and GraphSAGE), as shown in table 1. It is worth noting that the original LADIES and PLADIES were designed specifically for GraphSAGE. A comparison of BLISS on GAT against the original PLADIES (or LADIES) on GraphSAGE reveals a noticeable advantage for BLISS (e.g., Citeseer: 70.6% vs. 60.1%; Pubmed: 73.1% vs. 55.7%).

The results demonstrate superior F1-scores for BLISS, particularly with GAT models on Citeseer (70.6% vs. 68.3%) and Pubmed (73.1% vs. 71.8%). This advantage stems from its bandit-driven mechanism, which better adapts to evolving node importance, thereby reducing variance and improving generalization. The performance gains are most pronounced on smaller datasets (Cora, Citeseer, Pubmed) and on complex, heterogeneous graphs like Yelp, where BLISS effectively captures nuanced class relationships (52.9% vs. 50.2% with SAGE). On denser, more uniform graphs like Flickr and Reddit, the performance difference is minimal. fig. 2, fig. 3 summarizes the F1-scores (mean ± standard deviation) and loss for both samplers on GAT and GraphSAGE architectures.

These results validate our theoretical analysis: BLISS minimizes estimator variance by dynamically prioritizing informative neighbors, unlike PLADIES' static sampling which risks under-sampling critical nodes. The only noted exception was overfitting on the Yelp dataset with GAT for both samplers, which was unevaluated to maintain uniform experimental conditions.

4 Conclusion

In this work, we proposed Bandit-Based Layer Importance Sampling Strategy (BLISS), a layer-wise sampling method for scalable and accurate training of deep GNNs on large graphs. BLISS employs multi-armed bandits to dynamically select informative nodes per layer, balancing exploration of under-sampled regions with exploitation of valuable neighbors. This enables efficient message passing and improved scalability. We demonstrated its applicability to diverse GNNs, including GCNs and GATs, and presented an adaptation of PLADIES for GATs. Experiments show BLISS matches or exceeds state-of-the-art performance while remaining computationally efficient. Future directions include exploring advanced bandit algorithms (e.g., CMAB) and extending BLISS to domains such as GNN-augmented LLMs and vision tasks.

References

Muhammed Fatih Balin and Ümit Çatalyürek. Layer-neighbor sampling — defusing neighborhood explosion in gnns. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/51f9036d5e7ae822da8f6d4adda1fb39-Paper-Conference.pdf.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Yifan Chen, Tianning Xu, Dilek Hakkani-Tur, Di Jin, Yun Yang, and Ruoqing Zhu. Calibrate and debias layer-wise sampling for graph convolutional networks. *arXiv preprint arXiv:2206.00583*, 2022.

Zheng Chen, Ziyan Jiang, Fan Yang, Eunah Cho, Xing Fan, Xiaojiang Huang, Yanbin Lu, and Aram Galstyan. Graph meets llm: A novel approach to collaborative filtering for robust conversational understanding. *arXiv preprint arXiv:2305.14449*, 2023.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision gnn: An image is worth graph of nodes. *Advances in neural information processing systems*, 35:8291–8303, 2022.

Yan Han, Peihao Wang, Souvik Kundu, Ying Ding, and Zhangyang Wang. Vision hgnn: An image is more than a graph of nodes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19878–19888, 2023.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems*, 31, 2018.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Jaechang Lim, Seongok Ryu, Kyubyong Park, Yo Joong Choe, Jiyeon Ham, and Woo Youn Kim. Predicting drug–target interaction using a novel graph neural network with 3d structure-embedded graph representation. *Journal of chemical information and modeling*, 59(9):3981–3988, 2019.

Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *Advances in Neural Information Processing Systems*, 33:6878–6888, 2020.

Amil Merchant, Simon Batzner, Samuel S Schoenholz, Muratahan Aykol, Gowoon Cheon, and Ekin Dogus Cubuk. Scaling deep learning for materials discovery. *Nature*, 624(7990):80–85, 2023.

Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. Query-driven active surveying for collective classification. In *10th international workshop on mining and learning with graphs*, volume 8, page 1, 2012.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. *arXiv preprint arXiv:2310.13023*, 2023.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 950–958, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

Minji Yoon, Jing Yu Koh, Bryan Hooi, and Ruslan Salakhutdinov. Multimodal graph learning for generative tasks. *arXiv preprint arXiv:2310.07478*, 2023.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems*, 32, 2019.

A Tables

A.1 Notation Summary

Table 2: Summary of key notations used throughout the paper, including their descriptions and contexts. This table serves as a reference for understanding the mathematical formulations and algorithms presented in the paper.

Symbol	Description	Context
α_{ij}	Edge weight between node i and j	GNN
$\hat{h}_j$	Feature vector of sampled node j	GNN
$\mathcal{N}(i)$	Set of neighbors of node i	GNN
k	Sample size (number of neighbors)	Sampling
q_i	Probability distribution of sampling node from neighbors of i	Sampling
$\hat{\mu}_i$	Estimated representation for node i	Sampling
q_{ij}	Sampling probability of neighbor node j from node i,	Sampling
p_j	Sampling probability of a node j from all nodes of the current layer	Sampling
s	Index of the sample neighbor	Sampling
j_s	s-th sampled neighbor of node i	Sampling
w_{ij}	Edge weight between nodes i and j learned through the Bandit Algorithm	BLISS/Bandit
r_{ij}	Reward of edge e_{ij}	BLISS/Bandit
η	Learning rate for EXP3 algorithm	BLISS/Bandit
δ	Scaling factor for EXP3 algorithm	BLISS/Bandit
S_i^t	Set of sampled neighbors of node i at iteration t	BLISS/Bandit
c	Scaling factor in Poisson Sampling	PLADIES
ϵ	Tolerance used in Poisson Sampling	PLADIES
n_{ref}	Refinement factor used in Poisson Sampling	PLADIES

A.2 Dataset

Table 3: Summary of the datasets used in the experiments, including the number of nodes, edges, features, classes, and the split ratio for training, validation, and testing. This table provides an overview of the graph properties and complexity of each dataset, highlighting the diversity in scale and structure.

Dataset	# Classes	# Nodes	# Edges	# Features	# Train	# Validation	# Test
Cora	7	2,708	10,556	1,433	140	500	1,000
Citeseer	6	3,327	9,228	3,703	120	500	1,000
Pubmed	3	19,717	88,651	500	60	500	1,000
Flickr	7	89,250	899,756	500	44,625	22,312	22,313
Reddit	41	232,965	11,606,919	602	153,431	23,831	55,703
Yelp	100	716,847	13,954,819	300	537,635	107,527	71,685

Table 4: Experiment settings for each dataset, including batch size, fanout configuration, and the number of training steps.

Dataset	Batch Size	Fanouts	Steps
Citeseer	32	[512, 256, 128]	1000
Cora	32	[512, 256, 128]	1000
Flickr	256	[4096, 2048, 1024]	1000
Pubmed	32	[512, 256, 128]	1000
Reddit	256	[4096, 2048, 1024]	3000
Yelp	256	[4096, 2048, 1024]	10000

B Background

We denote a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of a set of nodes $\mathcal{V} = \{v_i\}_{i=1:N}$ and a set of edges $\mathcal{E} = \{e_{ij} | j \in \mathcal{N}_i\}_{i=1:N} \subseteq \mathcal{V} \times \mathcal{V}$, where N is the number of nodes, $\mathcal{N}_i$ denotes the set of neighbors of node v_i, and L is the number of layers.

Graph Neural Networks. GNNs operate on the principle of neural message passing Gilmer et al. [2017], where nodes iteratively aggregate information from their local neighborhoods. In a typical GNN, the embedding of node v_i at layer $l+1$ is computed from layer l as follows:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} W^{(l)} h_j^{(l)}\right) \tag{7}$$

where $W^{(l)}$ is a learnable weight matrix, $h_j^{(l)}$ is the node feature vector at layer l, and σ is a non-linear activation function. The term α_{ij} represents the aggregation coefficient, which varies depending on the GNN architecture (e.g., static in GCNs Kipf and Welling [2016] or dynamic in GATs Velickovic et al. [2017]).

Layer-wise Sampling.
Following Huang et al. [2018], eq. (7) can be written in expectation form:

$$h_i^{(l+1)} = \sigma_{W^{(l)}}\left(N(i)\, \mathbb{E}_{p_{ij}}\left[h_j^{(l)}\right]\right) \tag{8}$$

where $p_{ij} = p(v_j | v_i)$ is the probability of sampling v_j given v_i, and $\mathcal{N}(i) = \sum_j \alpha_{ij}$. To make the computation of eq. (8) tractable, the expectation $\mu_p(i) = \mathbb{E}_{p_{ij}}[h_j^{(l)}]$ can be approximated via Monte-Carlo sampling:

$$\hat{\mu}_p(i) = \frac{1}{n} \sum_{j=1}^{n} \hat{h}_j^{(l)}, \hat{v}_j \sim p_{ij} \tag{9}$$

eq. (9) defines *node-wise sampling*, where neighbors are recursively sampled for each node. While this reduces immediate computational load, the receptive field still grows exponentially with network depth d, leading to $O(n^d)$ dependencies in the input layer for deep networks. An alternative approach is to apply importance sampling to eq. (8), which forms the basis for *layer-wise sampling* methods:

$$h_i^{(l+1)} = \sigma_{W^{(l)}}\left(N(i)\, \mathbb{E}_{q_j}\left[\frac{p_{ij}}{q_j} h_j^{(l)}\right]\right) \tag{10}$$

where $q_j = q(v_j | v_1, ..., v_n)$ is the probability of sampling node v_j from the entire layer. We estimate the expectation $\mu_q(i)$ via Monte-Carlo sampling:

$$\hat{\mu}_q(i) = \frac{1}{n} \sum_{j=1}^{n} \frac{p_{ij}}{q_j} \hat{h}_j^{(l)}, \hat{v}_j \sim q_j \tag{11}$$

The embedding then becomes $h_i^{(l+1)} = \sigma_{W_{(l)}}\left(\mathcal{N}(i) \hat{\mu}_q(i)\right)$. Without loss of generality, following the setting from Liu et al. [2020], we assume $p_{ij} = \alpha_{ij}$ and normalize the probabilities such that $\mathcal{N}(i) = 1$. We denote $\mu_q(i)$ as μ_i for simplicity and ignore non-linearities. The goal of a layer-wise sampler is to approximate:

$$h_i^{(l+1)} = \hat{\mu}_i = \frac{1}{n} \sum_{j=1}^{n} \frac{\alpha_{ij}}{q_j} \hat{h}_j^{(l)}, \hat{v}_j \sim q_j \tag{12}$$

An effective estimator should minimize variance. The variance of the estimator in eq. (12) is:

$$\begin{aligned} \mathbb{V}(\hat{\mu}_i) &= \mathbb{E}\left[\left(\hat{\mu}_i - \mathbb{E}\left[\hat{\mu}_i\right]\right)^2\right] = \mathbb{E}\left[\left\|\hat{\mu}_i - \mu_i\right\|^2\right] \\ &= \mathbb{E}\left[\left\|\frac{\alpha_{ij}}{q_j} h_j^{(l)} - \sum_{s \in \mathcal{N}_i} \alpha_{sj} h_j^{(l)}\right\|^2\right] \end{aligned} \tag{13}$$

We seek $q_j^\star \geq 0$ that minimizes $\mathbb{V}(\hat{\mu}_i)$. The optimal sampling distribution is:

$$q_j^\star = \sqrt{\sum_i \left(\frac{\alpha_{ij} \left\|h_j^{(l)}\right\|_2}{\sum_{s \in \mathcal{N}_i} \alpha_{sj} \left\|h_j^{(l)}\right\|_2}\right)^2} \tag{14}$$

C Related Work

To address these challenges, efficient neighbor sampling techniques are crucial. These methods typically involve randomly selecting a fixed number of neighbors, sampling based on node features or importance scores, or employing adaptive strategies that learn optimal sampling during training. They can be broadly categorized into three groups: **1) Node-wise sampling** samples a subset of neighbors for each node. While this reduces immediate computations and memory usage, the recursive nature can introduce redundancy. Examples include GraphSAGE Hamilton et al. [2017], VR-GCN Chen et al. [2017], and BS-GNN Liu et al. [2020]. **2) Layer-wise sampling** jointly selects neighbors for all nodes at each layer, potentially offering better efficiency and capturing broader relationships than purely node-wise methods. However, it can introduce biases if certain graph parts are consistently under-sampled. Examples include FastGCN Chen et al. [2018], LADIES Zou et al. [2019], and LABOR Balin and Çatalyürek [2023]. See fig. 1 for a visual example. **3) Sub-graph sampling** focuses on smaller, self-contained induced subgraphs for message passing. While efficient, using the same subgraph across all layers risks losing global context. Examples include Cluster-GCN Chiang et al. [2019] and GraphSAINT Zeng et al. [2019].

C.1 GNN Architectures

Graph Convolutional Networks (GCNs) leverage a simplified convolution operation on the graph, aggregating information from a node's neighbors, and produces the normalized sum of them as in eq. (7) where σ is an activation function (ReLU for GCNs), $\mathcal{N}(i)$ is the set of its one-hop neighbors, $\alpha_{ij}^{(l)} = \frac{1}{c_{ij}}$, $c_{ij} = \sqrt{|\mathcal{N}(i)|}\sqrt{|\mathcal{N}(j)|}$, $W^{(l)}$ is the weight matrix for the l-th layer, $h_j^{(l)}$ denotes the node feature matrix at layer l. For **GraphSAGE** is $c_{ij} = |\mathcal{N}(i)|$. This equally weights the contributions from all neighbors.
Graph Attention Networks (GATs) address the equal contribution by introducing an attention mechanism that assigns learnable weights (α) to each neighbor based on their features, allowing the model to focus on the most relevant information:

$$e_{ij}^{(l)} = \text{LeakyReLU}(\vec{a}^{(l)^\top}(W^{(l)}h_i^{(l)}||W^{(l)}h_j^{(l)})) \tag{15}$$

this computes a pair-wise un-normalized attention score between two neighbors. It first concatenates the linear transformation of l-th layer embeddings of the two nodes, where $||$ denotes concatenation, then takes a dot product of it and a learnable weight vector $\vec{a}^{(l)}$, and applies a LeakyReLU in the end.

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k\in\mathcal{N}(i)}\exp(e_{ik}^{(l)})} \tag{16}$$

This difference allows GATs to capture more nuanced relationships within the graph than GCNs. However, Brody et al. [2021] argues that the original GAT uses a static attention mechanism due to the specific order of operations in eq. (15). While the weights depend on both nodes, this structure can limit the expressiveness of the attention calculation. GATv2 introduces a dynamic attention mechanism by modifying this order. This change allows the attention weights to depend on the features of both the sending node (neighbor) and the receiving node in a potentially more expressive way. The key difference lies in the order of operations. eq. (15) will be changed to:

$$e_{ij}^{(l)} = \vec{a}^{(l)^\top}\text{LeakyReLU}(W^{(l)}[h_i^{(l)}||h_j^{(l)}]) \tag{17}$$

C.2 Sampling Technique

This section reviews existing sampling techniques and discusses their limitations.
Layer-Dependent Importance Sampling (LADIES) Zou et al. [2019]: LADIES leverages layer-wise importance scores based on node features and graph structure to guide node selection. LADIES begins by selecting a subset of nodes in the upper layer. For each selected node, it constructs a bipartite subgraph of its immediate neighbors. It then calculates importance scores for these neighbors and samples a fixed number based on these scores. This process is repeated recursively for each layer. However, LADIES relies on pre-computed importance scores, which can be computationally expensive and may not adapt well to dynamic edge-weight changes. Additionally, LADIES employs sampling with replacement, which can be suboptimal as it may select the same node multiple times.

Algorithm 1 Sampling Procedure of LADIES

Require: Normalized edge weights α_{ij}; Batch Size b, Sample Number k;
1: Randomly sample a batch of b output nodes.
2: **for** $l \leftarrow L$ to 1 **do**
3: Calculate sampling probability for each node using p_j^l in eq. (18)
4: Sample k nodes in l-th layer using p_j^l.
5: Normalize the edge weights of the sampled nodes in the layer by eq. (19).
6: **end for**
7: **return** Modified edge weights $\tilde{\alpha}_{ij}$ and Sampled Nodes;

While LADIES suggests using α_{ij} values similar to GraphSAGE ($c_{ij} = |\mathcal{N}(i)|$), their implementation utilizes eq. (19) for normalization. Instead of directly feeding α_{ij} to the model, it is first used to calculate an importance score:

$$p_j^{(l)} = \frac{\pi_j^{(l)}}{\sum_j \pi_j^{(l)}} \text{ , where } \pi_j^{(l)} = \sum_{j \in \mathcal{N}_i^{(l)}} \alpha_{ij}^2 \tag{18}$$

The most important nodes are then sampled using $p_j^{(l)}$. Before passing these nodes to the model, the original α_{ij} is re-weighted by $p_j^{(l)}$ and normalized by dividing it over c_{ij} of the selected (union of sampled nodes and seed nodes) nodes. These new $\tilde{\alpha}_{ij}^{(l)}$ values are passed to the model at each layer for the selected points.

$$\tilde{\alpha}_{ij}^{(l)} = \frac{\alpha_{ij}/p_j^{(l)}}{\sum_j \left(\alpha_{ij}/p_j^{(l)}\right)}, \tilde{\alpha}_{ij}^{(l)} = \frac{\alpha_{ij}/p_j^{(l)}}{c_{ij}} \tag{19}$$

SKETCH Chen et al. [2022] proposed a fix for the sampling equation and the normalization of the edge weights. Instead of eq. (18), they suggested:

$$\pi_j^{(l)} = \sqrt{\sum_{j \in \mathcal{N}_i^{(l)}} \alpha_{ij}^2} \tag{20}$$

SKETCH uses α_{ij} based on the GCN model ($c_{ij} = \sqrt{|\mathcal{N}(i)|}\sqrt{|\mathcal{N}(j)|}$). They also suggested an alternative normalization for the edge weight instead of eq. (19):

$$\tilde{\alpha}_{ij}^{(l)} = \frac{\alpha_{ij}/p_j^{(l)}}{ns_j^{(l)}} \tag{21}$$

where $ns_j^{(l)}$ is the number of sampled nodes for node i at layer l.

Layer-Neighbor Sampling (LABOR) Balin and Çatalyürek [2023]: The LABOR sampler combines layer-based and node-based sampling. It introduces a per-node hyperparameter to estimate the expected number of sampled neighbors, enabling correlated sampling decisions among vertices. This hyperparameter and the sampling probabilities are optimized to sample the fewest vertices in an unbiased manner.

The paper also introduced PLADIES (Poisson LADIES), which employs Poisson sampling to achieve unbiased estimation with reduced variance. PLADIES assigns each node j in the neighborhood of source nodes S (denoted $N(S)$) a sampling probability $p_j \in [0, 1]$ such that $\sum_{j \in N(S)} p_j = k$, where k is the desired sample size. A node j is then sampled if a random number $\phi_j \sim U(0, 1)$ satisfies $\phi_j \leq p_j$. PLADIES achieves this unbiased estimation in linear time, in contrast to the quadratic complexity of some debiasing methods Chen et al. [2022]. Notably, its variance converges to 0 if all $p_j = 1$, highlighting its effectiveness.

Bandit Samplers Liu et al. [2020]: Bandit Samplers frame the optimization of sampling variance as an adversarial bandit problem, where rewards depend on evolving node embeddings and model

weights. While node-wise bandit sampling is effective, selecting neighbors individually can lead to redundancy and may not capture long-range dependencies efficiently. This highlights the importance of extending to layer-wise sampling.

Their method employs a multi-armed bandit framework to learn a sampling distribution q_i^t for each node v_i at each training step t. The algorithm initializes a uniform sampling distribution. During training, it samples k neighbors for each node based on q_i^t, computes rewards based on GNN performance, and updates the distribution using an algorithm like EXP3. This process prioritizes informative neighbors to improve training efficiency. Our work builds upon this foundation by applying bandit principles to the layer-wise sampling paradigm.

D Complexity, Variance and Runtime

D.1 BLISS complexity and variance analysis

Table 5: Summary of memory complexity, time complexity, and variance for Full-Batch, GraphSAGE, LADIES, and BLISS methods. This table provides a theoretical comparison of the computational and statistical properties of each method, emphasizing BLISS's ability to minimize variance while maintaining scalability.

Methods	**Memory Complexity**	**Time Complexity**	**Variance**
Full-Batch	$O\left(L\|V\|K + LK^2\right)$	$O\left(L\|\|A\|\|K + L\|V\|K^2\right)$	0
GraphSage	$O\left(bKs_{node}^{L-1} + LK^2\right)$	$O\left(bKs_{node}^{L} + bK^2s_{node}^{L-1}\right)$	$O\left(D\phi\|P\|_F^2/(\|V\|s_{node})\right)$
LADIES	$O\left(LKs_{layer} + LK^2\right)$	$O\left(LKs_{layer}^2 + LK^2s_{layer}\right)$	$O\left(\phi\|P\|_F^2\bar{V}(b)/(\|V\|s_{layer})\right)$
BLISS	$O\left(L\|E\| + LKs_{\text{layer}} + LK^2\right)$	$O(L\|E\| + LKs_{\text{layer}}^2 + LK^2s_{\text{layer}})$	$O\left(\phi\|P\|_F^2\bar{V}(b)/\|V\|s_{\text{layer}}(1-\eta)\right)$

D.1.1 Memory Complexity

- $O(L|E|)$: Stores bandit weights w_{ij} for all edges across L layers.
- $O(LKs_{\text{layer}})$: Stores embeddings for s_{layer} sampled nodes per layer (K-dimensional).
- $O(LK^2)$: Stores L weight matrices $\mathbf{W}^{(l)} \in \mathbb{R}^{K\times K}$.

D.1.2 Time Complexity

- $O(L|E|)$: Bandit weight updates (EXP3) over all edges in L layers.
- $O(LKs_{\text{layer}}^2)$: Importance score computation for s_{layer} nodes per layer.
- $O(LK^2s_{\text{layer}})$: Message passing and aggregation for s_{layer} nodes.

D.1.3 Variance

- Key Difference from LADIES: The $(1-\eta)^{-1}$ term accounts for exploration in bandit sampling.
- Derivation: Minimizing eq. (13) with bandit-optimized q_j (eq. (6)) introduces the η-dependent denominator.

D.2 PLADIES complexity and variance analysis

PLADIES (Poisson LADIES) shares identical complexity terms with LADIES. The differences between them is that PLADIES uses Poisson sampling (variable-size, unbiased) instead of fixed-size sampling, and PLADIES reduces empirical variance but retains the same asymptotic bound.

In the table 5, PLADIES is grouped under LADIES since their theoretical complexities are identical. BLISS explicitly diverges due to bandit overhead and adaptive exploration.

D.3 Training time

The reported time in table 6 measures per-iteration training time - the wall-clock time taken to execute one training step. The code for BLISS is not optimized (using naive for loops in the current

implementation) and the comparison might not be reasonable, but for the sake of the having a clearer image about the performance. The time is also averaged over 5 runs per experiment.

Table 6: Average training time per iteration (in seconds) for BLISS and PLADIES samplers across six datasets. The table highlights the computational efficiency of both samplers, with BLISS incurring slightly higher overhead due to its dynamic bandit-based sampling mechanism, and the naive loop implementation.

Dataset	Sampler	Time		Dataset	Sampler	Time	
		GAT	SAGE			GAT	SAGE
Citeseer	BLISS	0.065 ± 0.001	0.059 ± 0.002	Pubmed	BLISS	0.722 ± 0.008	0.690 ± 0.007
	PLADIES	0.055 ± 0.001	0.051 ± 0.002		PLADIES	0.667 ± 0.008	0.627 ± 0.007
Cora	BLISS	0.066 ± 0.001	0.058 ± 0.001	Reddit	BLISS	0.207 ± 0.003	0.165 ± 0.007
	PLADIES	0.054 ± 0.001	0.049 ± 0.001		PLADIES	0.156 ± 0.002	0.110 ± 0.003
Flickr	BLISS	0.086 ± 0.002	0.080 ± 0.002	Yelp	BLISS	0.129 ± 0.003	0.122 ± 0.004
	PLADIES	0.073 ± 0.002	0.063 ± 0.002		PLADIES	0.110 ± 0.002	0.102 ± 0.002

E Algorithms

E.1 BLISS

Algorithm 2 BLISS Algorithm

Require: Graph G, Sample size k, Bandit learning rate η, Steps T, Number of layers L

Initialize $w_{ij} = 1$ if $j \in \mathcal{N}_i$ else 0
for $t = 1$ to T **do**
  for $l = L$ to 1 **do** ▷ Top-down layers
    Calculate sampling distribution q_{ij}, using eq. (6)
    Calculate node sampling probability p_j, using eq. (1)
    Pass the p_j to algorithm 4
    Sample k nodes for the current layer based on p_j.
  end for
  Run forward pass of GNN
  Get the updated node embeddings h_j from eq. (2) and rewards r_{ij} using eq. (3)
  Update wights w_{ij} using EXP3 in algorithm 5
end for

E.2 Iterative Thinning Poisson Sampler

Algorithm 3 Iterative Thinning Poisson Sampler

Require: Node probabilities p_j, sample size k, tolerance ϵ, refinement factor n_{ref}

Initialize scaling factor $c = 1.0$
for $i = 1$ to n_{ref} **do**
  Adjust probabilities: $S = \sum \min(p_j \cdot c, 1)$
  if $\min(S, k)/\max(S, k) \geq \epsilon$ **then**
    break
  end if
  Update scaling factor: $c = c \cdot k/S$
end for
return c

E.3 PLADIES

Algorithm 4 Poisson Sampling with Skip Connections

Require: Input subgraph G_{sub}, seed nodes V_s, edge probabilities α_{ij}, sample size k, tolerance ϵ, refinement factor n_{ref}

Compute node probabilities p_j based on edge weights w_{ij} from eq. (18).
if $|V_{\text{sub}}| \leq k$ **then**
    return $p_j \leftarrow 1 \; \forall j \in V_{\text{sub}}$ ▷ Include all nodes
end if
$c \leftarrow \text{ITERATIVETHINNINGPOISSONSAMPLER}(p_j, k, \epsilon, n_{\text{ref}})$ ▷ From algorithm 3
$V_{\text{skip}} \leftarrow \{j \mid j \in V_s \cap V_{\text{sub}}\}$ ▷ Identify seed nodes in subgraph
for $j \in V_{\text{sub}}$ **do**
    if $j \in V_{\text{skip}}$ **then**
        $p_j \leftarrow \infty$ ▷ Ensure seed nodes are always selected
    else
        $p_j \leftarrow \min(p_j \cdot c, 1)$ ▷ Clip and scale probabilities
    end if
end for
return $p_j \; \forall j \in V_{\text{sub}}$

E.4 EXP3

Algorithm 5 EXP3

Require: Neighbor size n, Sample size k, Bandit learning rate η, Number of layers L

for $l = L$ to 1 **do**
    Calculate $\alpha_{ij} = \sum_{j \in S_i} q_{ij} \cdot \frac{\tilde{\alpha}_{ij}}{\sum_{j \in S_i} \tilde{\alpha}_{ij}}$
    Calculate estimated rewards $\hat{r}_{ij} = \frac{r_{ij}}{p_i}$
    Update weights $w_{ij} = w_{ij} \exp(\frac{\delta r_{ij}}{n_i p_i})$
end for

F Plots

The advantages of BLISS are particularly pronounced in smaller datasets (Citeseer, Cora, Pubmed) and highly heterogeneous graphs like Yelp (100 classes) with SAGE. For Yelp, BLISS achieves a test F1-score 52.9% (SAGE), while PLADIES lags at 50.2%. The bandit mechanism likely captures nuanced class relationships more effectively in such complex settings. In contrast, Flickr and Reddit exhibit minimal differences between the samplers, possibly due to their dense connectivity and uniform class distributions, which reduce the impact of adaptive sampling.

GAT models generally benefit more from BLISS than SAGE. For example, on Cora, BLISS achieves a test F1-score of 81.3% (GAT) compared to 80.9% for PLADIES, while SAGE shows narrower margins 79.5% vs. 77.2%. This aligns with our hypothesis that attention mechanisms, which dynamically weigh neighbor contributions, synergize well with BLISS's reward-driven sampling. SAGE's uniform aggregation is less sensitive to neighbor selection, though BLISS still improves its performance.

Despite larger fanouts and batch sizes for Flickr, Reddit, and Yelp (table 4), BLISS maintains computational efficiency. Reddit's test F1-scores 94.9% for BLISS vs. 95.0% for PLADIES, highlight that both samplers scale effectively to massive graphs, though BLISS's adaptive policy incurs negligible overhead. The higher step counts for Reddit (3,000) and Yelp (10,000) reflect their size but do not compromise BLISS's stability, as evidenced by low standard deviations.

The Yelp dataset with GAT presented a challenge for both samplers, showing overfitting (fig. 3). While early stopping or hyperparameter adjustments could potentially alleviate this, they were not added here to preserve uniform experimental conditions across all datasets.

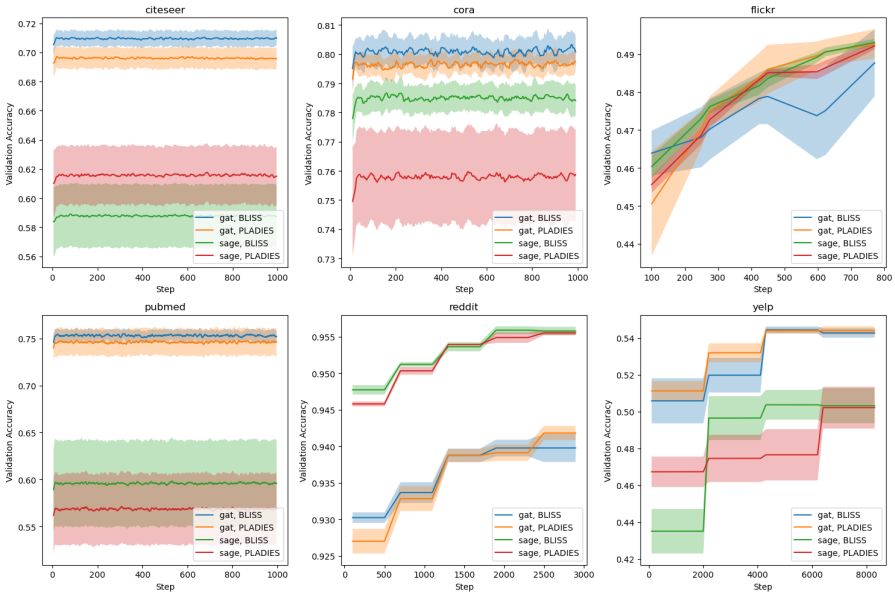

Figure 2: Validation Accuracy across six datasets (Citeseer, Cora, Pubmed, Flickr, Yelp, and Reddit) for BLISS and PLADIES samplers using Graph Attention Networks (GAT) and GraphSAGE (SAGE) architectures. The figure highlights the performance trends during training averaged over 5 runs. The shaded regions represent the standard deviation across runs.

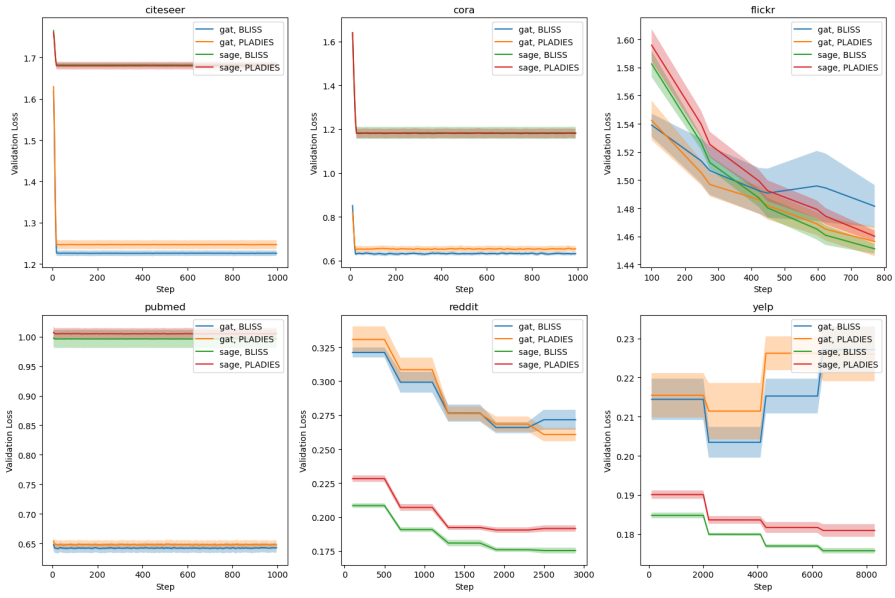

Figure 3: Validation Loss for the same datasets and models as in fig. 2. The figure illustrates the loss trends during training, averaged over 5 runs. The shaded regions represent the standard deviation across runs.