E2G-Net: Enhancing Efficiency in Graph Neural Networks With Early-Exit Branches

Haseena Rahmath P, Kuldeep Chaurasia, and Anika

***Abstract*—Graph Neural Networks (GNNs) are effective for learning on graph-structured data but often suffer from high inference costs, particularly in deeper architectures. Standard GNNs employ a single-exit design, processing all inputs through the entire network regardless of their complexity—resulting in unnecessary computation for simpler instances. This paper introduces E2G-Net, a multi-exit GNN architecture that inserts early-exit branches at intermediate layers to enable instance-adaptive inference. A Bayesian Optimization (BO)-based policy determines the optimal exit criterion and threshold at each branch, optimizing the trade-off between accuracy and efficiency. E2G-Net is evaluated using GCN and GAT backbones on ten node classification benchmarks spanning homophilic, heterophilic, and large-scale graphs. It achieves up to 3.7× inference speedup (Cornell) and over 45% FLOPs reduction (OGBN-Arxiv), while preserving and often improving classification accuracy across datasets. These results demonstrate E2G-Net's scalability and efficiency for real-world graph inference.**

***Index Terms*—Early-exit, multi-exit, fast inference, graph neural networks.**

I. Introduction

GRAPH Neural Networks (GNNs) have emerged as powerful models for learning on graph-structured data, achieving strong performance across various domains [1], [2], [3]. However, most GNNs remain shallow due to challenges such as overfitting, vanishing gradients, over-smoothing, and over-squashing in deeper architectures [4], [5], [6]. These issues lead to performance degradation and hinder scalability. While deeper GNNs can capture richer structural dependencies [7], they often suffer from increased latency and computational cost, limiting their applicability in real-time or resource-constrained settings.

In Convolutional Neural Networks (CNNs), techniques such as compression and quantization have been widely adopted to reduce inference cost. Another effective strategy is the early-exit mechanism, which inserts intermediate exits to allow confident inputs to terminate early [8]. This avoids redundant computation by halting inference once sufficient confidence is reached. As shown in Fig. 1, early exits enable input-adaptive processing: easy samples exit at E_1, medium at E_2, and only the hardest reach the final exit, FE. While early-exit mechanisms have demonstrated success in CNNs and RNNs, they remain underexplored in GNNs due to their unique structural and computational characteristics. Unlike CNNs, which operate on grid-like data, GNNs process irregular, non-Euclidean structures with dynamic message passing, heterogeneous neighborhoods, and often high-dimensional intermediate features. These properties complicate the integration of early-exit branches and underscore the need for GNN-specific early-exit frameworks.

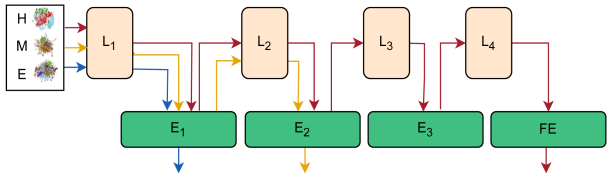


Fig. 1. Early-exit mechanism: E, M, and H denote easy, medium, and hard samples. L_1–L_4 are intermediate layers; E_1–E_3 are early exits, and FE is the final exit. Colored paths indicate sample flow.

To address this gap, this study proposes E2G-Net, a multi-exit GNN architecture that integrates early-exit branches directly into standard GNN backbones such as GCN [9] and GAT [10]. E2G-Net enables instance-adaptive inference by dynamically adjusting depth: most inputs exit early based on prediction confidence, while only difficult samples traverse deeper layers. This design reduces inference time and computation without compromising accuracy, making E2G-Net well-suited for latency-sensitive and resource-constrained scenarios.

The contributions are summarized as follows:

1) *Inference Acceleration:* E2G-Net enables adaptive inference through early-exit branches, allowing most inputs to be classified without traversing the full network. This significantly reduces inference time and energy consumption while preserving accuracy.
2) *Implicit Optimization Benefits:* The early-exit design helps mitigate common depth-related challenges such as over-smoothing, over-squashing, vanishing gradients, and overfitting. Joint training across all exits introduces auxiliary gradient signals and implicit regularization that improve overall model performance.
3) *Bayesian Optimization-Based Exit Policy:* E2G-Net employs a BO-guided, data-driven strategy to determine the optimal confidence metric and threshold for each

Received 21 April 2025; accepted 9 May 2025. Date of publication 5 June 2025; date of current version 25 November 2025. *(Corresponding author: Haseena Rahmath P.)*

Haseena Rahmath P and Kuldeep Chaurasia are with the School of Computer Science Engineering and Technology, Bennett University, 201310 Greater Noida, India (e-mail: haseenarahmath@gmail.com; kchaurasia.iitr@gmail.com).

Anika is with the Department of Computer Science Engineering, School of ICT, Gautam Buddha University, 201312 Greater Noida, India (e-mail: anikagupta2010@gmail.com).

This article has supplementary downloadable material available at https://doi.org/10.1109/TETCI.2025.3573240, provided by the authors.

Recommended for acceptance by M. Gong.

Digital Object Identifier 10.1109/TETCI.2025.3573240

exit branch. This approach systematically improves exit decision quality while maintaining minimal inference-time overhead.

The remainder of the paper is structured as follows. Section II reviews relevant literature. Section III outlines the research questions. Section IV presents the E2G-Net architecture. Sections V and VI detail the experimental setup and results. Section VII compares E2G-Net to state-of-the-art models. Section VIII discusses limitations, and Section IX concludes the paper.

II. Literature Review

This section reviews the evolution of GNN models, examines early-exit mechanisms in neural networks, and discusses recent efforts to improve GNN scalability, robustness, and depth.

A. Graph Neural Networks

GNNs are widely used to model graph-structured data, effectively capturing complex node relationships. Early developments began with Scarselli et al. [11] and spectral extensions by Bruna et al. [12] and Defferrard et al. [13]. Kipf and Welling introduced the Graph Convolutional Network (GCN) [9], which simplifies spectral filtering using localized approximations. GAT [10] further enhanced GNN expressiveness via masked self-attention. Spatial approaches such as GraphSAGE [14] introduced sampling-based aggregation for scalable embedding generation.

While GCN and GAT remain foundational, numerous variants target node classification, link prediction, and graph classification [1], [2], [3]. Traditional GNNs assume homophily, but real-world graphs often exhibit heterophily [15]. To address this, models like H2GCN [16], Geom-GCN [17], GPRGNN [18], and GloGNN++[19] improve cross-class aggregation. PEGFAN [20] uses permutation-equivariant graph framelets for robust multi-scale learning. Meanwhile, studies show that even GCNs can generalize under proper training [15], [21]. Other works focus on efficiency: GCN-RW [22] uses random filters and least-squares loss to reduce training cost, while HAQJSK [23] introduces quantum kernels with hierarchical alignment for accurate, permutation-invariant graph classification.

B. Challenges in GNNs

Despite significant progress, GNNs continue to face a wide range of challenges that limit their scalability and applicability in real-world scenarios. These include: (i) scalability issues due to exponential neighborhood expansion, (ii) sensitivity to noisy labels, (iii) limitations in inductive generalization, and (iv) degradation in deeper architectures caused by over-smoothing and over-squashing. Recent literature has comprehensively addressed these challenges through various strategies [6], [47], [48], [49], [50], [51], [52], [53], [54].

To address scalability, sampling-based methods and mini-batching [54] reduce memory and compute costs, while propagation schemes like PPRGo [47] and GBP [48] improve efficiency on large graphs. GNN robustness under label noise is enhanced through contrastive learning [51], homophily-based filtering, and reweighting strategies used in CR-GNN [53], UnionNET [52], and SuLD-GCN [51]. For inductive learning, models such as GraphIMOS [54] employ subgraph-based training to enable generalization to unseen nodes without retraining.

Depth-related issues persist in deep GNNs, where over-smoothing causes node embeddings to become indistinct, and over-squashing compresses long-range dependencies. Kipf and Welling [9] and Veličković et al. [10] observed that performance typically declines beyond a few layers. Recent efforts to overcome these effects include residual and identity mapping [4], [7], normalization techniques [5], graph rewiring [6], [50], [55], and maximization-based propagation strategies [49].

While these advances address foundational challenges, this work focuses specifically on improving computational efficiency through faster and adaptive inference. E2G-Net introduces early-exit branches that allow confident predictions to terminate at intermediate layers, reducing unnecessary computation and avoiding depth-induced degradation. This adaptive strategy enhances scalability and runtime efficiency, making E2G-Net particularly suited for real-time and resource-constrained GNN applications.

C. Early-Exit Mechanism in Neural Networks

Early-exit mechanisms reduce inference cost by enabling dynamic computation—terminating the forward pass once a confident prediction is reached. First introduced by Teerapittayanon et al. [8], this approach integrates auxiliary exit branches at intermediate layers of deep neural networks (DNNs), allowing simpler inputs to exit early while more complex instances propagate deeper. A conceptual illustration is provided in Fig. 1. This mechanism has since been widely applied to various architectures, including CNNs [8], [24], [27], [28], [29], [30], [31], [38], RNNs [43], [44], and Transformer-based models [32], [34], [35], [41]. Table I summarizes representative early-exit strategies, their backbone architectures, and the exit policies they employ.

The effectiveness of early-exit models largely depends on the exit policy—that is, the criterion for deciding when to halt inference. Most prior works employ static, heuristic-based policies that compare a confidence score (e.g., maximum softmax probability [27], [28], [29], [32], [36], [37], [39], [40] or entropy [8], [24], [41]) to a manually tuned threshold. Beyond heuristics, some works explore more advanced strategies, including optimization-based [25], [38] and reinforcement learning-based [30], [31], [44] approaches. Others propose prediction consistency [34], [35], gating mechanisms [43], or reliance on backbone accuracy [45]. However, most approaches still depend on fixed or brute-force-tuned parameters. Prior studies have shown that exit thresholds significantly influence both performance and efficiency [8], [39], [40], highlighting the need for data-driven, systematically optimized exit policies.

D. Early-Exit Architectures in GNNs

While early-exit mechanisms have been extensively explored in CNNs, RNNs, and Transformers, their integration into GNNs

TABLE I
SUMMARY OF EARLY-EXIT NEURAL NETWORKS WITH UNDERLYING BACKBONE MODELS AND EXIT POLICIES

Reference	Backbone	Exit Policy
Teerapittayanon et al. [8] (2016), Scardapane et al. [24] (2020)	AlexNet	Entropy-Threshold
Zeng et al. [25] (2019)	AlexNet	Optimizer Algorithm
Wang et al. [26] (2021)	ResNet	Entropy-Threshold
Berestizshevsky et al. [27] (2019), Vashist et al. [28] (2022), Leontiadis et al. [29] (2021)	ResNet	MaxProb-Threshold
Guan et al. [30] (2017), Wu et al. [31] (2018)	ResNet	Reinforcement Agent
Xin et al. [32] (2020), Bajpai et al. [33] (2024)	BERT	MaxProb-Threshold
Zhou et al. [34] (2020), Xin et al. [35] (2021)	BERT	Prediction Constancy
Li et al. [36] (2019), Jiang et al. [37] (2020)	MSDNet	MaxProb-Threshold
Scardapane et al. [24] (2020)	VGG	Entropy-Threshold
Zhao et al. [38] (2021)	VGG	Optimizer Algorithm
Leontiadis et al. [29] (2021), Pacheco et al. [39] (2021), Rahmath et al. [40] (2023)	MobileNetV2	MaxProb-Threshold
Xin et al. [41] (2020), Liu et al. [42], (2023)	RoBERTa	Entropy-Threshold
Shen et al. [43] (2017)	RNN/LSTM/GRU	Termination Gate
Fan et al. [44] (2018)	LSTM/GRU	Reinforcement Agent
Shao et al. [45] (2021)	DGCNN	Backbone accuracy
Yang et al. [46] (2024)	MSDNet	Entropy-Threshold
Proposed*	GCN, GAT	BO-based

remains largely unexplored. One notable attempt is Branchy-GNN [45], which applies early exits to a hybrid GNN-CNN architecture for point cloud classification. However, Branchy-GNN introduces exit branches only within the CNN layers (e.g., Conv2d) while treating the GNN component as a fixed, non-exitable encoder. These CNN-based exits are trained independently and are designed primarily for device-edge co-inference in vision tasks. As such, Branchy-GNN is fundamentally CNN-driven and lacks applicability to core GNN tasks such as node classification and link prediction.

To address this gap, we introduce E2G-Net—a general-purpose, GNN-native early-exit framework that embeds exit branches directly within the GNN backbone. E2G-Net enables instance-adaptive inference by dynamically adjusting the depth of computation based on input complexity. It supports standard GNN architectures like GCN and GAT, and is applicable to a wide range of tasks including node classification, link prediction, and graph classification. Unlike previous hybrid approaches, E2G-Net jointly trains the backbone and all exit branches within a unified objective, ensuring coherent feature representations and gradient flow across depths.

Beyond accelerating inference, E2G-Net provides a novel strategy for addressing depth-related limitations in GNNs. While existing approaches aim to mitigate over-smoothing and over-squashing using deeper architectures with residual connections [4], normalization techniques [5], or graph rewiring [55], E2G-Net takes a fundamentally different approach—it prevents unnecessary depth traversal altogether. By allowing confident samples to exit early, it actively reduces over-propagation, improving both efficiency and representation stability.

Furthermore, most existing early-exit methods rely on manually selected thresholds and fixed confidence metrics, which are often sensitive to tuning and dataset characteristics. In contrast,

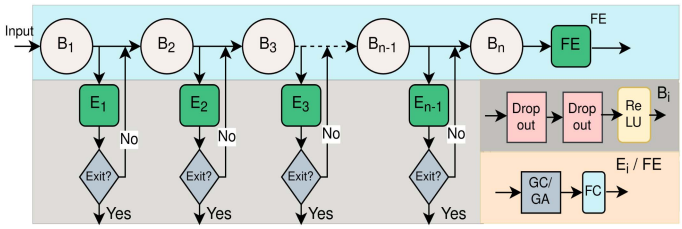


Fig. 2. Early-exit GNN architecture with early-exit branches integrated into a GNN backbone. Each block B_i contains dropout, a graph convolution (GC) or graph attention (GA) layer, and a ReLU. Exits E_i (i = 1 to n-1) and final exit FE consist of a GC/GA layer and a fully connected layer for classification.

E2G-Net employs a BO-based policy, applied post-training to determine the most suitable exit criterion and its threshold for each exit branch. This policy is optimized per dataset to balance accuracy and efficiency and remains static during inference—offering a principled, data-driven alternative to heuristic thresholding while eliminating the runtime overhead associated with dynamic exit strategies.

III. RESEARCH QUESTIONS

This study investigates the following research questions to evaluate the performance and efficiency of E2G-Net:

- *RQ1:* Can E2G-Net make accurate predictions at early-exit branches?
- *RQ2:* How effectively does E2G-Net improve inference speed and efficiency?
- *RQ3:* How does E2G-Net scale to deeper GNNs in terms of performance and computational overhead?
- *RQ4:* Does E2G-Net enable efficient exit policy optimization?
- *RQ5:* Can E2G-Net scale to large graphs while maintaining efficiency and accuracy?

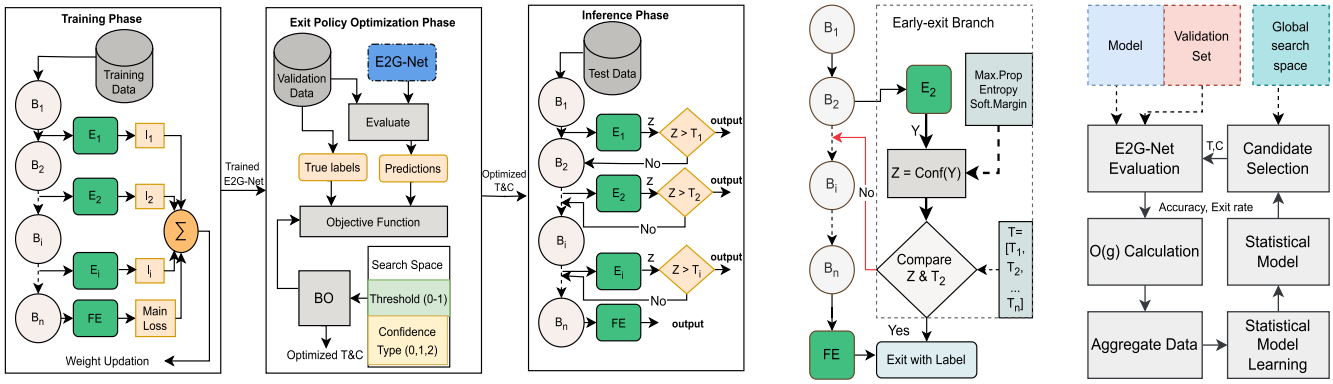
IV. E2G-NET: EARLY-EXIT GRAPH NEURAL NETWORK

This section presents the proposed E2G-Net model, detailing its architecture, training procedure, inference mechanism, and the optimization of exit policies.

A. Architecture

The architecture of E2G-Net, illustrated in Fig. 2, extends standard GNN backbones—such as GCN and GAT—by inserting early-exit branches after each graph convolution or attention layer. These models are selected as backbone to demonstrate E2G-Net's generality across different GNN paradigms. The backbone comprises a sequence of neural blocks (B_i), each containing a dropout layer, a graph convolution or attention layer, and a nonlinear activation function (ReLU or ELU). A final output layer (FE) serves as the full-depth prediction head. After each block B_i, an early-exit branch E_i is attached, consisting of a lightweight convolution layer followed by a fully connected layer, forming a side classifier.

Each early-exit branch produces a prediction and includes a confidence estimator, which may use one of several criteria: maximum softmax probability, entropy, or softmax-margin (i.e.,



(a) E2G-Net Learning and Inference Process (b) Inference at E_2 (c) BO-based Tuning

Fig. 3. B_1 to B_n, and FE represent backbone blocks, E_1 to E_{n-1} are early-exit branches. Y is the classification label, and Z is the confidence score of E_i in predicting Y. Each E_i has a threshold T_i used for exit decisions.

the gap between the top-2 predicted class probabilities). The estimated confidence is compared against a threshold T_i to determine whether to terminate inference at E_i or continue to deeper layers. To automate exit decision design, a BO-based strategy is employed post-training to determine, for each exit, the optimal confidence criterion and its corresponding threshold. This results in a static, dataset-specific exit policy defined as $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$, along with the selected confidence metric for each branch. The final policy minimizes inference cost while maintaining high predictive performance.

B. Training E2G-Net

E2G-Net extends the standard GNN architecture by incorporating early-exit branches, necessitating a unified training strategy. The entire model—including the backbone and all exit branches—is trained jointly to ensure consistent learning across all components. Fig. 3(a) outlines the overall pipeline, comprising three phases: training, exit policy optimization, and inference.

During training, E2G-Net treats the backbone and exit branches as a single optimization unit. All input samples propagate through the full network during training, regardless of whether they would exit early at inference time. This ensures that each exit branch receives gradient updates and contributes to the loss, thereby avoiding the "dead-end" issue observed in methods like BranchyNet [8], where early-exiting samples may prevent later layers from being sufficiently trained.

The model is optimized using a weighted sum of cross-entropy losses from each exit branch and the final output layer. The total loss $\mathcal{L}$ is defined as:

$$\mathcal{L} = \sum_{n=1}^{N+1} W_n L_n, \tag{1}$$

where N is the number of exits, L_n is the cross-entropy at the n-th exit, and W_n is its corresponding weight. Each L_n is given by:

$$L_n = -\frac{1}{|C|} \sum_{c \in C} y_c \log \hat{y}_c, \tag{2}$$

Algorithm 1: E2G-Net Training.

Require: E2G-Net with N exits, feature matrix X, adjacency matrix adj
Ensure: Trained E2G-Net
1: Initialize: $\mathcal{L} = 0$, $n = 1$, MaxEpoch $= 200$
2: **for** $i = 1$ to MaxEpoch **do**
3: **for** layer in model.layers() **do**
4: loss $=$ layer.forward(X, adj)
5: **if** layer.type $==$ 'Exit' **then**
6: $w = \frac{\alpha \cdot n}{\sum_{k=1}^{N} k}$
7: $n = n + 1$
8: **else**
9: $w = 1$
10: **end if**
11: $\mathcal{L} += w \cdot$ loss
12: **end for**
13: model.backward($\mathcal{L}$)
14: **end for**

with $\hat{y}_c = \frac{\exp(z_c)}{\sum_{j \in C} \exp(z_j)}$ representing the softmax output, y_c is the one-hot encoded label, and C the label set. The loss weights are computed as: $W_n = \frac{\alpha \cdot n}{\sum_{k=1}^{N} k}$, where α is a tunable hyperparameter that controls the relative contribution of each exit. Higher α values emphasize early exits, improving their accuracy at the potential cost of regularization pressure on deeper branches. In this study, weights ranged from 0.1 to 1, increasing linearly across exits.

Algorithm 1 outlines the joint training process. During the forward pass, each exit branch computes its loss and adds it to the total loss using its corresponding weight (Steps 4–11). Backpropagation is then applied using the combined loss to update the entire model (Step 13).

C. Inference With E2G-Net

Once trained, E2G-Net performs inference using the process described in Algorithm 2, enabling early classification of input instances through intermediate exit branches. Starting from the

Algorithm 2: Inference using E2G-Net.

Input: Input sample x; trained E2G-Net with N exit branches
Output: Predicted class label $\hat{y}$
1: **for** $i = 1$ to N **do**
2: $x = \mathrm{B}_i(x)$
3: $y_i, z_i = \mathrm{E}_i(x)$
4: $\hat{y} = \arg\max(y_i)$
5: **if** $z_i \geq T_i$ **then**
6: **return** $\hat{y}$
7: **end if**
8: **end for**
9: **return** $\hat{y}$

first exit, each input x propagates sequentially through the backbone and exits until reaching the final layer, which always produces a prediction.

At each exit branch E_i, the model computes a predicted label y_i and an associated confidence score z_i based on the selected confidence metric (e.g., softmax probability or entropy). If z_i exceeds the threshold T_i, the model terminates inference early and returns the predicted class $\hat{y} = \arg\max(y_i)$. If no exit meets its threshold, the input proceeds to the final output layer. This inference flow is illustrated in Fig. 3(b). The threshold vector $\mathbf{T} = \{T_1, T_2, \ldots, T_N\}$ and the confidence metric for each exit are determined during the Bayesian Optimization phase, as detailed in Section IV-D. In Algorithm 2, B_i denotes the layers in the backbone GNN up to E_i, and E_i denotes the i^{th} exit branch.

D. Exit-Policy Optimization

This study uses Bayesian Optimization to determine the optimal confidence measure and threshold for each exit branch, balancing inference efficiency and accuracy.

Problem Formulation: Let T_i denote the confidence threshold at exit branch E_i, with bounds $T_i \in (l, u)$. For instance, when using maximum softmax probability, $T_i \in (0, 1)$. Three confidence measures are supported: maximum probability (code 0), entropy (1), and softmax-margin (2). The search space for thresholds is denoted by P, and that for confidence types by M. The joint search space is $G = P \times M$.

The objective is to find a configuration $g \in G$ that maximizes the following performance-efficiency trade-off:

$$\mathcal{O}(\text{E2G-Net}, \lambda) = \sum_{i=1}^{N} (\lambda A_i R_i + (1 - \lambda) F_i) + \mathcal{A}, \quad (3)$$

where A_i is the accuracy at exit E_i, R_i is the proportion of samples exiting at E_i, F_i is the cumulative inference cost up to E_i, $\mathcal{A}$ is the overall classification accuracy, and $\lambda \in [0, 1]$ controls the trade-off between early-exit efficiency and global accuracy.

BO is used to select both $T_i \in P$ and the confidence metric $m_i \in M$ per exit. The search seeks $g^* = \arg\max_{g \in G} \mathcal{O}(g)$ using iterative evaluations on the validation set.

Algorithm 3: Bayesian Optimization for Exit Policy Selection.

Require: Validation set $\mathcal{X}_{val}$, cost profile F, search space G of thresholds and confidence types
Ensure: Optimized exit thresholds $\{T_1, \ldots, T_N\}$ and confidence metric m
1: Initialize $\mathcal{X}_0$ with random samples from G
2: $i = 0$
3: **repeat**
4: Fit GP model: $\mathcal{M}_i = \text{Fit\_GP}(\mathcal{X}_i)$
5: Select next candidate: $g_{i+1} = \arg\max_{g \in G} \mathcal{AF}(\mathcal{M}_i, g)$
6: Evaluate objective: $\mathcal{O}(g_{i+1})$
7: Update dataset: $\mathcal{X}_{i+1} = \mathcal{X}_i \cup \{(g_{i+1}, \mathcal{O}(g_{i+1}))\}$
8: $i \leftarrow i + 1$
9: **until** convergence
10: **return** $g^* = \arg\max_{g_i} \mathcal{O}(g_i)$

Bayesian Optimization Framework: BO models the objective $\mathcal{O}(g)$ as a Gaussian Process (GP), $\mathcal{M}(g) \sim \mathcal{GP}(\mu(g), k(g, g'))$, where μ is the mean function and k is a covariance kernel. The kernel is defined as $k(g, g') = k_0 \phi(\|g - g'\|)$, where ϕ is a radial basis function (RBF), and k_0 controls prior variance. The next candidate g_{i+1} is selected by maximizing the Upper Confidence Bound (UCB) acquisition function:

$$\mathcal{AF}(g) = \mu_i(g) + \sqrt{k_{i+1}} \cdot \sigma_i(g), \quad (4)$$

where $\mu_i(g)$ and $\sigma_i(g)$ are the posterior mean and standard deviation from the GP at iteration i, and k_{i+1} adjusts the exploration–exploitation trade-off. The optimization iterates until convergence. At each step, BO:

1. Selects $g_{i+1} = \arg\max_{g \in G} \mathcal{AF}(g)$,
2. Evaluates $\mathcal{O}(g_{i+1})$ on the validation set,
3. Updates the dataset $\mathcal{X}_{i+1} = \mathcal{X}_i \cup \{(g_{i+1}, \mathcal{O}(g_{i+1}))\}$,
4. Refits the GP model $\mathcal{M}_{i+1}$.

Fig. 3(c) illustrates this optimization loop, where BO iteratively refines the exit policy by balancing predicted utility and uncertainty. Algorithm 3 summarizes the full procedure. For clarity, we provide a summary of key mathematical notations in Appendix A, available online.

V. Experiments

This section outlines the experimental protocol used to evaluate E2G-Net, including the datasets, baseline models, evaluation metrics, and experimental setup.

A. Dataset

E2G-Net is evaluated on ten benchmark datasets for the node classification task, covering both homophilic and heterophilic graph structures. All datasets are sourced from the PyTorch Geometric library. The homophilic category includes citation networks such as Cora [56], PubMed [57], and CiteSeer [58], as well as Amazon co-purchase networks: Computers and Photo [59], [60]. The heterophilic group comprises Wikipedia

TABLE II
PROPERTIES AND STATISTICS OF EVALUATION DATASETS

Datasets	#Nodes	#Edges	#Features	H(G)	#Classes
Cora	2,708	5,278	1,433	0.825	7
Pubmed	19,717	44,324	500	0.792	3
Citeseer	3,327	4,552	3,703	0.718	6
Computers	13,752	245,861	745	0.802	10
Photo	7,650	119,081	767	0.849	8
Actor	7,600	26,659	932	0.215	5
Squirrel	5,201	198,353	2,089	0.217	5
Chameleon	2,277	31,371	2,325	0.247	5
Texas	183	279	1,703	0.057	5
Cornell	183	277	1,703	0.301	5
OGBN-Arxiv	169,343	1,166,243	128	0.780	40

hyperlink graphs (Chameleon and Squirrel) [61], the Actor co-occurrence network [62], and WebKB3 webpage graphs (Texas and Cornell) [17]. In addition, we include OGBN-Arxiv dataset from the Open Graph Benchmark [63] to assess scalability to large-scale graphs. Dataset statistics are provided in Table II, including node/edge counts and the homophily ratio $\mathrm{H}(G)$, which quantifies the fraction of edges connecting nodes with the same class label [16]. For consistency, we adopt standardized train/validation/test splits. Homophilic datasets use the public splits from [9], [60], while heterophilic datasets follow the protocol in [17], enabling fair comparisons across all baseline models. For the large-scale OGBN-Arxiv dataset, we use the official split provided by the Open Graph For the large-scale OGBN-Arxiv dataset, we use the official split provided by the Open Graph Benchmark [63].

B. Baseline Models and Evaluation Metric

We evaluate E2G-Net using two popular GNN architectures—GCN [9] and GAT [10]—as backbone models. The standard GCN and GAT are two-layer models (i.e., depth 2^i with $i = 1$). We extend these architectures to deeper variants by increasing i, and use $i = 3$ (i.e., 8 layers) as the default configuration across most experiments. These models are denoted as GCN_8 and GAT_8, where the subscript indicates the total number of layers. This value provides a balanced trade-off between representational capacity and training complexity. However, the choice of i is flexible and can be scaled for deeper models; in Section VI, we explore depths up to $i = 6$ (i.e., 64 layers) to analyze E2G-Net's scalability.

E2G-Net integrates early-exit branches into GCN_8 and GAT_8, resulting in the early-exit models EEGCN and EEGAT, respectively. These are evaluated against several baselines: (1) standard shallow GNNs (2-layer GCN and GAT), (2) deeper GNNs without early exits (GCN_8 and GAT_8), (3) oversmoothing-resistant GNNs ($\mathrm{GCN}_{\mathrm{pn}}$ and $\mathrm{GAT}_{\mathrm{pn}}$), which incorporate PairNorm [5] into the 8-layer models, and (4) state-of-the-art models, including GloGNN and GloGNN++ [19], GPRGNN [18], and DualNet GNN [64]. We report node classification accuracy, averaged over five runs, with standard deviation to ensure statistical reliability. These comparisons evaluate E2G-Net's improvements in classification accuracy, inference speed, and computational efficiency across a range of GNN architectures.

C. Experimental Setup

All experiments are conducted on a machine equipped with a 2.60 GHz CPU, 32 GB RAM, and an NVIDIA GeForce RTX GPU. The implementation is written in Python using the PyTorch library for both E2G-Net and all baseline models. Models are trained for 500 epochs using the Adam optimizer with a learning rate of 0.01. E2G-Net consists of six neural blocks and five early-exit branches. Each neural block includes a dropout layer (dropout rate 0.06), a graph convolution or attention layer with 64 hidden units, and a ReLU or ELU activation function. Each early-exit branch contains a graph convolution or attention layer followed by a fully connected layer for classification.

VI. RESULTS AND DISCUSSION

RQ1: *Can E2G-Net make accurate predictions at early-exit branches?*

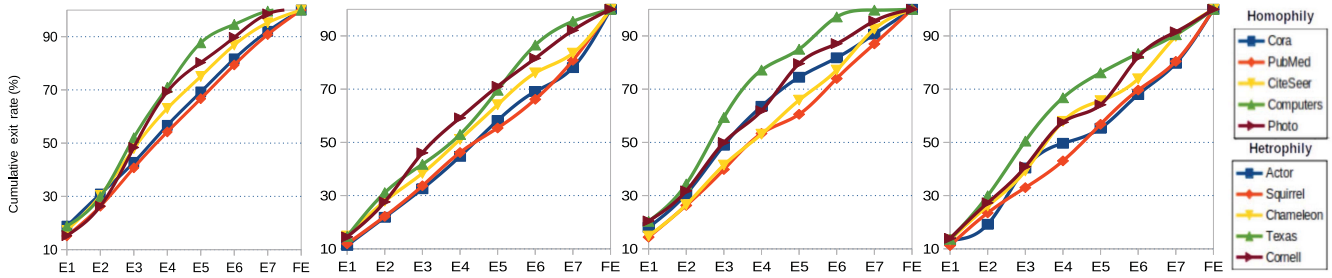
The effectiveness of E2G-Net's early-exit predictions is demonstrated by the cumulative exit rates and branch-wise accuracy results shown in Figure 4 and Table III. The cumulative exit curves reveal a steady increase across branches, indicating progressive feature refinement. High exit rates at intermediate branches, combined with strong classification accuracy, suggest that a substantial proportion of inputs can be confidently predicted early—thereby avoiding unnecessary full-depth inference. For example, on the Computers dataset, around 50% of inputs exit at E_3 with over 96% accuracy. On average, 80% of samples exit before the final branch, achieving accuracies of 85.57% for EEGCN and 79.58% for EEGAT. This balance between early exit rates and predictive accuracy remains consistent across both homophilic and heterophilic datasets, demonstrating E2G-Net's robustness and adaptability. These results affirm E2G-Net's ability to deliver accurate early predictions, enabling significant inference savings without sacrificing classification performance.

RQ2: *How effectively does E2G-Net improve inference speed and efficiency?*

E2G-Net accelerates GNN inference by enabling early exits for a majority of inputs, thus avoiding full-depth propagation. Figure 5 illustrates how computational cost—measured in FLOPs—progressively increases across exit branches, confirming that early exits offer substantial savings for simpler inputs. Figure 6 reports the average GPU inference time per sample, showing consistent reductions across all datasets when using E2G-Net versus its backbone models. Quantitative results in Table IV highlight these gains: EEGCN achieves up to 3.73× speedup, while EEGAT yields up to 1.79× reduction in inference time. These results demonstrate that E2G-Net significantly improves inference efficiency by dynamically adjusting computation to input complexity, making it highly suitable for latency-sensitive and resource-constrained environments.

RQ3: *How does E2G-Net scale to deeper GNNs in terms of performance and computational overhead?*

To assess E2G-Net's scalability, we conduct a depth-wise analysis focusing on model accuracy and computational cost. Specifically, we evaluate configurations with increasing depth



(a) EEGCN–Homophilic datasets (b) EEGCN – Heterophilic datasets (c) EEGAT – Homophilic datasets (d) EEGAT – Heterophilic datasets

Fig. 4. Exit rate distribution of E2G-Net (EEGCN and EEGAT) across datasets. Most samples exit early, confirming the effectiveness of instance-adaptive inference in reducing full-depth computation.

TABLE III
CLASSIFICATION ACCURACY (%) AT EACH EARLY-EXIT BRANCH (E_1–E_7) AND FINAL EXIT (FE) IN E2G-NET (EEGCN, EEGAT) ACROSS DATASETS

Dataset	EEGCN Accuracy (%)								EEGAT Accuracy (%)							
	E1	E2	E3	E4	E5	E6	E7	FE	E1	E2	E3	E4	E5	E6	E7	FE
Cora	94.92	94.24	95.03	95.31	95.54	96.51	97.21	97.91	81.94	83.26	81.93	81.78	95.99	91.47	91.08	90.70
Pubmed	85.13	86.72	88.40	87.11	88.20	89.57	90.72	92.25	81.00	81.51	82.31	80.54	82.72	87.83	80.16	74.78
Citeseer	85.44	84.81	85.49	86.55	86.41	87.73	88.35	89.36	82.97	84.07	84.82	82.46	92.27	88.41	82.02	91.62
Computers	96.22	97.29	97.50	98.60	99.20	98.01	99.00	99.17	94.22	96.22	95.96	97.29	97.29	96.50	97.29	98.60
Photo	96.57	96.94	97.38	96.81	96.39	94.03	99.08	97.68	93.28	93.86	92.35	93.55	95.27	94.27	96.12	97.92
Actor	36.09	35.82	35.92	36.77	36.01	35.91	36.87	24.24	34.09	36.09	35.82	35.92	36.77	36.01	37.23	24.24
Squirrel	75.18	77.27	79.31	78.79	82.35	83.71	82.35	59.27	35.50	38.50	41.38	43.08	48.00	55.56	61.54	34.50
Chameleon	71.15	73.96	72.86	77.42	77.49	79.14	78.65	71.97	51.69	59.00	63.75	64.01	64.36	63.76	65.96	69.38
Texas	83.61	90.91	91.12	94.65	95.55	97.16	94.65	97.95	83.82	85.49	91.29	93.93	93.99	94.54	95.52	96.36
Cornell	81.30	83.65	84.87	86.85	87.50	89.54	88.84	99.91	80.29	81.58	82.82	83.58	85.24	90.91	88.89	77.78

Results demonstrate that high-confidence predictions can be made early, enabling efficient inference.

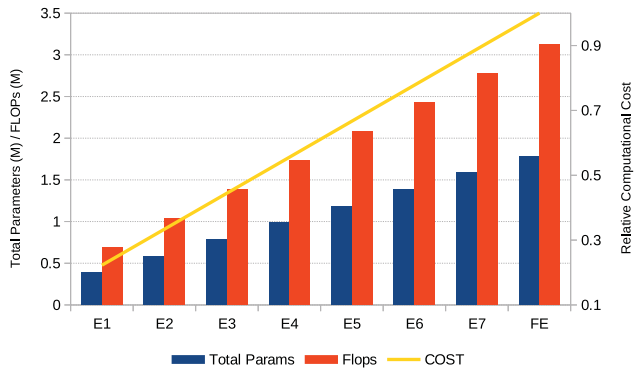


Fig. 5. FLOPs, total parameters, and relative computational cost of each early-exit branch in E2G-Net (EEGCN), computed on the Cora dataset. Relative cost is defined as the ratio of FLOPs at each exit to that of the full model. FLOPs and parameters are measured in millions.

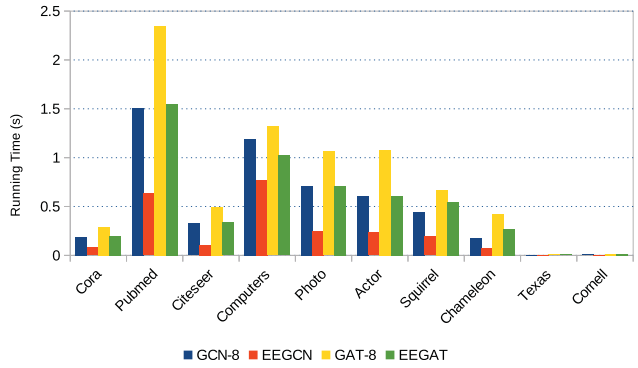


Fig. 6. Average GPU inference time per sample (ms) for E2G-Net vs. backbone models. E2G-Net consistently reduces latency by dynamically adapting depth based on input complexity.

TABLE IV
INFERENCE TIME GAIN OF E2G-NET (EEGCN, EEGAT) OVER BACKBONE MODELS ACROSS DATASETS

Homophilic Datasets					
	Cora	Pubmed	Citeseer	Computers	Photo
EEGCN	2.36x	2.38x	3.16x	1.54x	2.8x
EEGAT	1.48x	1.52x	1.46x	1.29x	1.49x
Heterophilic Datasets					
	Actor	Squirrel	Chameleon	Texas	Cornell
EEGCN	1.6x	1.3x	1.47x	1.1x	3.73x
EEGAT	1.79x	1.22x	1.57x	1.09x	1.03x

$L = 2^i$ ($i = 2$ to 6), up to 64 layers, and report overall accuracy, FLOPs, and average executed layers.

Computational Complexity: Figure 7 illustrates that FLOPs increase rapidly with depth in standard GNNs due to exponential neighborhood expansion. The theoretical complexity of an L-layer GNN is $O(Nd^L)$ [9], [14], where N is the number of nodes and d is the average node degree. Since all nodes propagate through every layer, deeper models incur significant computational cost. E2G-Net addresses this by allowing inputs to exit at depths proportional to their complexity, effectively reducing the average executed depth to $L_{\text{avg}} \ll L$. This leads to a reduced complexity of $O(Nd^{L_{\text{avg}}})$, even in deep configurations. For instance, in 64-layer models, most samples exit after only 8–16 layers—consistent with the 8-layer core architecture—highlighted in Fig. 7, confirming E2G-Net's scalability and efficiency.

Depth Robustness and Oversmoothing Mitigation: Increasing GNN depth often causes performance degradation due to compounding effects such as over-smoothing, vanishing gradients, and overfitting. To evaluate this, we compare E2G-Net against

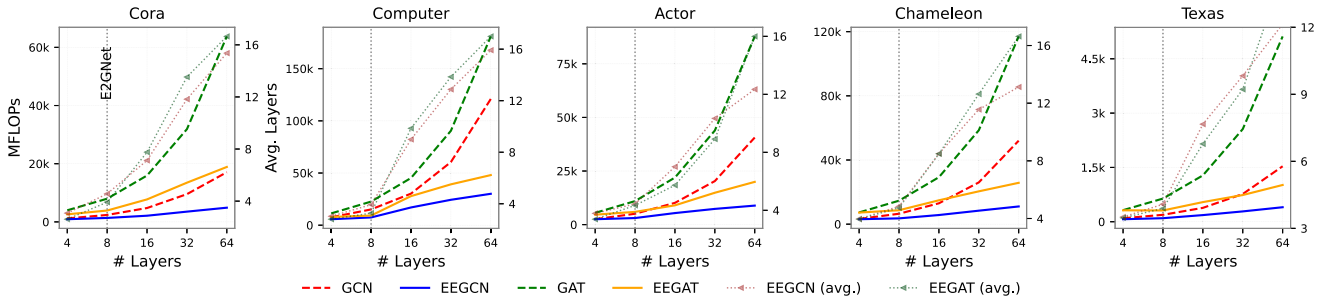


Fig. 7. FLOPs and average executed layers across depths (4–64) for GCN, GAT, and our E2G-Net variants (EEGCN, EEGAT). FLOPs are shown with solid/dashed lines; average executed layers are shown with dotted lines and triangle markers. The vertical dotted line marks the 8-layer E2G-Net baseline.

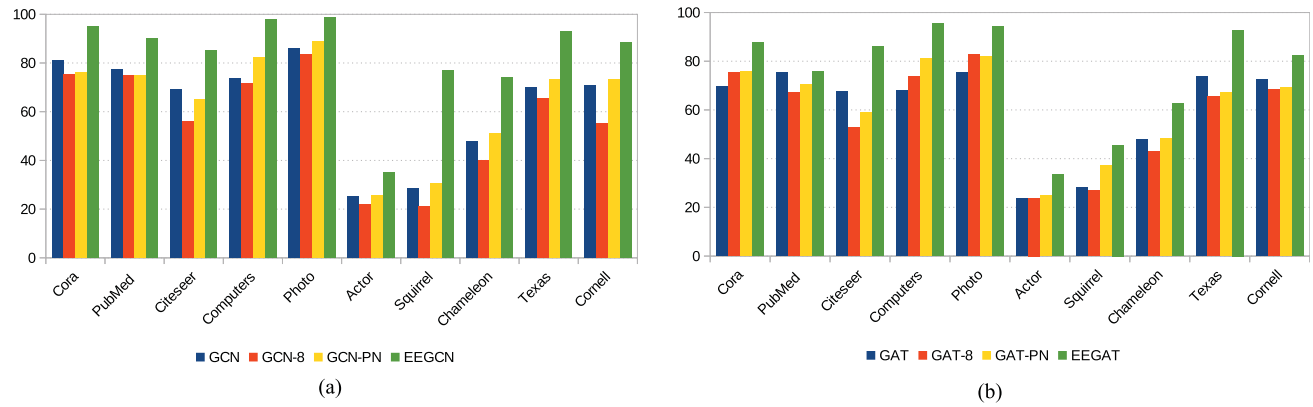


Fig. 8. Classification accuracy comparison across shallow, deep, oversmoothing-resistant, and our E2G-Net variants (EEGCN, EEGAT) on benchmark datasets.

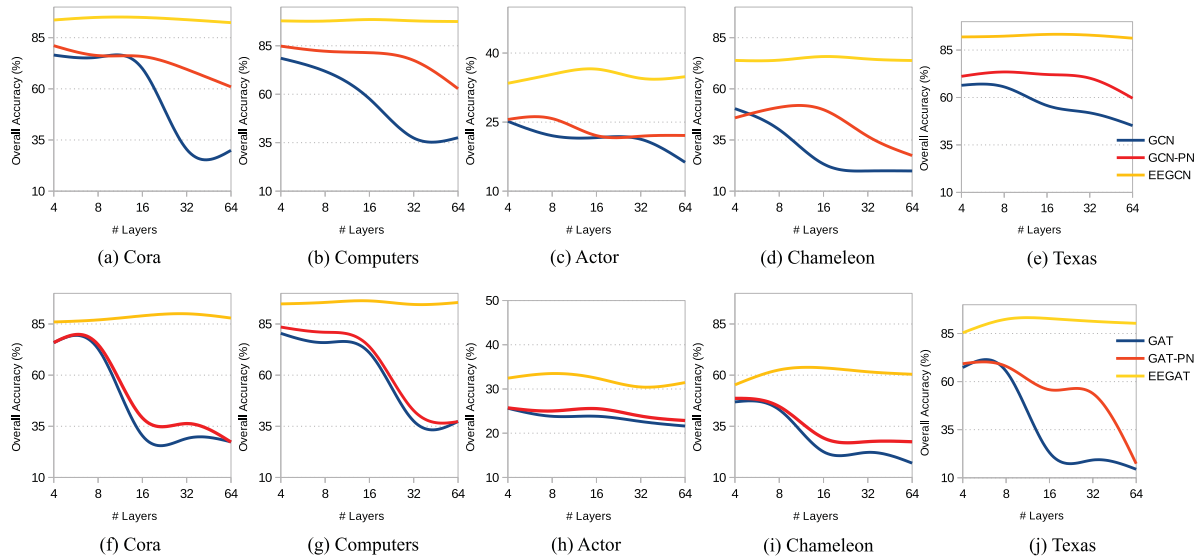


Fig. 9. Classification accuracy vs. model depth for standard GCN, oversmoothing-resistant GCN$_{pn}$, and E2G-Net (EEGCN variant) in the top row, and GAT, GAT$_{pn}$, and E2G-Net (EEGAT variant) in the bottom row, across five representative datasets.

8-layer GCN/GAT backbones and their oversmoothing-resistant variants using PairNorm [5]. As shown in Fig. 8(a) and (b), GCN$_8$ and GAT$_8$ degrade with depth, while GCN$_{\text{pn}}$ improves accuracy across datasets, and GAT$_{\text{pn}}$ yields modest gains, especially on heterophilic graphs.

In contrast, E2G-Net consistently outperforms all baselines, including those enhanced with PairNorm. For instance, EEGCN shows up to 41.29% improvement over GCN and up to 39.62% over GCN$_{\text{pn}}$, while EEGAT exceeds GAT and GAT$_{\text{pn}}$ by up to 27%. Fig. 9 reinforces this trend: while other models plateau or

TABLE V
BRUTE-FORCE VS BO-BASED OPTIMIZATION: BO JOINTLY TUNES METRIC AND THRESHOLD WITH 40× LOWER COST

Method	Configs Explored	Exit Scope	Time/Exit
Brute-force [8]	161,050	Threshold (fixed metric)	28 min
BO (proposed)	100	Metric + Threshold	36 sec

TABLE VI
EXIT-WISE AND OVERALL PERFORMANCE OF E2G-NET (EEGCN, EEGAT) ON OGBN-ARXIV

Exit	EEGCN		EEGAT	
	Exit Rate	Accuracy	Exit Rate	Accuracy
E1	13.89	76.46	9.38	74.62
E2	10.05	76.72	11.41	76.94
E3	11.47	77.81	8.74	76.45
E4	9.97	75.95	7.90	75.99
E5	8.70	76.79	9.04	75.15
E6	8.56	76.79	12.55	76.82
E7	11.46	77.67	9.08	77.51
FE	25.89	68.95	31.90	64.23
Overall Acc.	74.84		72.43	
Time Gain	1.88×		1.70×	
FLOPs Red.	46.86%		41.30%	
Avg. Layers	4.91		4.56	

Metrics include exit rate, accuracy, and inference time gain and FLOPs reduction relative to the respective backbone models (GCN_8 and GAT_8).

degrade beyond 16–32 layers, E2G-Net maintains strong performance across depths. This robustness stems from E2G-Net's early-exit design, which adaptively limits the number of layers traversed by easier instances, thereby mitigating the cumulative effects of over-smoothing. Additionally, joint optimization of a weighted loss across all exits introduces implicit regularization, while intermediate branches offer early gradient signals that help counter vanishing gradients and enhance representation learning in initial layers.

RQ4: *Does E2G-Net enable efficient exit policy optimization?*

The effectiveness of E2G-Net's BO-based strategy is evaluated against the traditional brute-force method [8] for determining exit policies. As shown in Table V, the brute-force approach evaluates 161,050 entropy threshold combinations, requiring approximately 28 minutes per exit branch. In contrast, E2G-Net's BO-based optimization converges within just 100 iterations and completes in 36 seconds. This efficiency is achieved by leveraging probabilistic modeling to intelligently explore the parameter space and identify optimal thresholds. Additionally, unlike fixed-confidence methods, which rely solely on manually tuned entropy or softmax thresholds, E2G-Net automatically selects both the best confidence metric and the corresponding threshold for each branch. Overall, E2G-Net's BO-based policy significantly reduces optimization time while enabling more flexible, data-driven, and accurate early-exit decisions.

RQ5: *Can E2G-Net scale to large graphs while maintaining efficiency and accuracy?*

Scaling GNNs to large graphs poses significant challenges due to the memory and computational demands of message passing across millions of nodes and edges. To evaluate the scalability of E2G-Net, we test it on the OGBN-Arxiv dataset, which includes over 160,000 nodes and 1 million edges—a representative benchmark for large, sparse citation networks [63]. As shown in Table VI, EEGCN achieves 74.84% accuracy, surpassing the best reported performance of the standard GCN (73.53% ± 0.12) [65]. EEGAT attains 72.43%, delivering comparable performance to the reported GAT result (73.30% ± 0.18) [65], while substantially reducing inference time and FLOPs. Notably, EEGCN and EEGAT achieve 1.88× and 1.70× speedup, and 46.86% and 41.30% FLOPs reduction, respectively. These results demonstrate that E2G-Net scales effectively to large graphs, achieving strong accuracy with markedly reduced computational overhead—underscoring its suitability for real-world, resource-constrained graph inference scenarios. A detailed ablation study analyzing the contribution of each core design component (e.g., joint training, BO policy, loss weighting) is provided in Appendix B, available online.

VII. ACCURACY COMPARISON WITH STATE-OF-THE-ART

Table VII presents a comparison of E2G-Net's classification accuracy against state-of-the-art models, including GloGNN, GloGNN++, GPRGNN, and Dual-Net GNN. Accuracy scores are sourced from the respective original publications. For consistency, GPRGNN's results are referenced from [64] due to differing train–test splits. E2G-Net consistently outperforms standard GCN and GAT models across all datasets, with particularly notable improvements in deeper configurations. It matches or exceeds the performance of state-of-the-art baselines on datasets such as Cora, Citeseer, Photo, Computers, Chameleon, Texas, and Cornell. On PubMed, E2G-Net achieves parity with the best reported results. While EEGAT exhibits some variability, it remains competitive, especially on datasets like Citeseer and Texas. Heterophilic benchmarks—Actor, Squirrel, and Chameleon—pose significant challenges for all models. Although E2G-Net improves performance on these graphs, overall accuracy remains modest, underscoring the inherent difficulty of learning in heterophilic settings. For instance, on the Actor dataset, the highest reported accuracy is 37.70%, with EEGCN and EEGAT achieving 35.28% and 33.49%, respectively. Overall, these results demonstrate that integrating early-exit strategies into standard GNNs enhances classification performance while remaining competitive with state-of-the-art models across a broad range of graph learning tasks.

VIII. LIMITATIONS

This study presents comprehensive evaluations of E2G-Net across both small- and large-scale graphs, including OGBN-Arxiv, covering accuracy, exit behavior, FLOPs, and runtime. While these results validate its scalability, future work could explore additional large-scale or dynamic graphs to assess generalization under diverse conditions. E2G-Net is implemented with GCN and GAT backbones, representing spectral and attention-based models. Evaluating its adaptability to other architectures—such as GraphSAGE, GIN, or heterophilic GNNs—remains a promising direction. Most experiments use a fixed 8-layer depth for consistency, though deeper models (up to 64 layers) were analyzed in Section VI. future work could investigate more diverse architectures or task-specific tuning. Finally, the current experiments focus on node classification tasks. Extending E2G-Net to edge and subgraph-level tasks may broaden its applicability across graph learning domains.

TABLE VII
E2G-NET VS. STATE-OF-THE-ART MODELS ON EXPERIMENTAL DATASETS: MEAN CLASSIFICATION ACCURACY (%) ± STANDARD DEVIATION OVER 5 TRIALS

Dataset	GCN [9]	GAT [10]	GloGNN [19]	GloGNN++ [19]	GPRGNN [18]	Dual-Net GNN [64]	E2G-Net (Ours)	
							EEGCN	EEGAT
Cora	80.97±1.32	69.69±2.69	88.31±1.13	88.33±1.09	88.99±0.95	87.77±1.16	**94.94±0.64**	87.66±0.29
Pubmed	77.28±1.5	75.20±1.44	89.62±0.35	89.24±0.39	85.07±0.40	**89.64±0.43**	88.96±0.64	75.81±0.04
Citeseer	69.38±1.17	67.45±1.03	77.41±1.65	77.22±1.78	77.08±1.63	77.15±1.58	85.33±0.31	**86.06±0.54**
Computers	71.06±2.18	68.03±1.89	NA	NA	82.90±0.37	NA	**97.75±0.04**	95.43±0.08
Photo	86.08±3.19	75.42±2.05	NA	NA	91.93±0.26	NA	**98.92±0.03**	94.1±0.08
Actor	25.27±0.72	23.57±0.84	37.35±1.30	**37.70±1.40**	36.04±0.96	37.29±0.80	35.28±0.25	33.49±0.29
Squirrel	28.71±1.7	28.16±1.03	69.78±2.42	71.21±1.84	49.03±1.28	**78.46±0.66**	75.86±0.62	45.63±0.29
Chameleon	48±1.59	47.75±2.1	57.54±1.39	57.88±1.76	66.47±2.47	73.97±1.89	**74.02±0.59**	61.8±0.79
Texas	70.19±2.36	73.65±2.62	84.32±4.15	84.05±4.90	86.49±4.83	87.57.64±3.24	91.64±3.08	**92.94±1.25**
Cornell	70.89±4.12	72.44±6.1	83.51±4.26	85.95±5.10	81.89±6.17	88.11±5.69	**88.61±2.99**	82.3±1.8

Best scores are in bold and the second-best scores are underlined.

IX. CONCLUSION

This paper introduces E2G-Net, a GNN architecture that enables faster and adaptive inference by integrating early-exit branches. These intermediate exits allow confident samples to terminate inference early, reducing unnecessary computation. A BO-based policy selects optimal thresholds and confidence measures for each branch, balancing accuracy and efficiency. E2G-Net also addresses depth-related challenges such as oversmoothing by leveraging instance-level complexity for dynamic depth adjustment. Extensive experiments on ten standard benchmarks and the large-scale OGBN-Arxiv dataset demonstrate that E2G-Net significantly reduces inference cost and improves accuracy over non-early-exit counterparts, while remaining competitive with state-of-the-art GNN models. These results underscore E2G-Net's suitability for deployment in latency-sensitive or resource-limited environments, providing an effective balance of performance, efficiency, and scalability across diverse graph learning scenarios.

REFERENCES

[1] S. Job, X. Tao, T. Cai, L. Li, H. Xie, and J. Yong, "Towards causal classification: A comprehensive study on graph neural networks," 2024, *arXiv:2401.15444*.

[2] C. Li and D. Goldwasser, "Encoding social information with graph convolutional networks forpolitical perspective detection in news media," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 2594–2604.

[3] J. Li et al., "Predicting path failure in time-evolving graphs," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1279–1289.

[4] G. Li, M. Muller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 9267–9276.

[5] L. Zhao and L. Akoglu, "PairNorm: Tackling oversmoothing in GNNs," in *Proc. 8th Int. Conf. Learn. Representations*, Apr. 2020. [Online]. Available: https://openreview.net/forum?id=rkecl1rtwB

[6] J. H. Giraldo, K. Skianis, T. Bouwmans, and F. D. Malliaros, "On the trade-off between over-smoothing and over-squashing in deep graph neural networks," in *Proc. 32nd ACM Int. Conf. Inf. Knowl. Manage.*, 2023, pp. 566–576.

[7] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 1725–1735.

[8] S. Teerapittayanon, B. McDanel, and H. -T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[10] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[11] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[12] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds., 2014.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, vol. 29, pp. 3837–3845.

[14] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30, pp. 1025–1035.

[15] S. Luan et al., "Revisiting heterophily for graph neural networks," in *Proc. Adv. Neural Inf Process. Syst.*, 2022, vol. 35, pp. 1362–1375.

[16] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 7793–7804.

[17] H. Pei, B. Wei, K. C. Chang, Y. Lei, and B. Yang, "Geom-GCN: Geometric graph convolutional networks," in *Proc. 8th Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1e2agrFvS

[18] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *Proc. 9th Int. Conf. Learn. Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=n6jl7fLxrP

[19] X. Li et al., "Finding global homophily in graph neural networks when meeting heterophily," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 13242–13256.

[20] J. Li, R. Zheng, H. Feng, M. Li, and X. Zhuang, "Permutation equivariant graph framelets for heterophilous graph learning," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 35, no. 9, pp. 11634–11648, Sep. 2024.

[21] Y. Ma, X. Liu, N. Shah, and J. Tang, "Is homophily a necessity for graph neural networks?," in *Proc. 10th Int. Conf. Learn. Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=ucASPPD9GKN

[22] C. Huang, M. Li, F. Cao, H. Fujita, Z. Li, and X. Wu, "Are graph convolutional networks with random weights feasible?," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 2751–2768, Mar. 2023.

[23] L. Bai et al., "HAQJSK: Hierarchical-aligned quantum Jensen-Shannon kernels for graph classification," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 11, pp. 6370–6384, Nov. 2024.

[24] S. Scardapane, D. Comminiello, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Differentiable branching in deep networks for fast inference," in *Proc. 2020 IEEE Int. Conf. Acoust., Speech Signal Process.*, 2020, pp. 4167–4171.

[25] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial Internet of Things," *IEEE Netw.*, vol. 33, no. 5, pp. 96–103, Sep./Oct. 2019.

[26] M. Wang, L. He, J. Lin, and Z. Wang, "Rethinking adaptive computing: Building a unified model complexity-reduction framework with adversarial robustness," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 33, no. 4, pp. 1803–1810, Apr. 2022.

[27] K. Berestizshevsky and G. Even, "Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence," in *Proc. Int. Conf. Artif. Neural Networks*, 2019, pp. 306–320.

[28] A. Vashist, S. V. Vidya Shanmugham, A. Ganguly, and S. M. P D, "DQN based exit selection in multi-exit deep neural networks for applications targeting situation awareness," in *Proc. 2022 IEEE Int. Conf. Consum. Electron.*, 2022, pp. 1–6.

[29] I. Leontiadis, S. Laskaridis, S. I. Venieris, and N. D. Lane, "It's always personal: Using early exits for efficient on-device CNN personalisation," in *Proc. 22nd Int. Workshop Mobile Comput. Syst. Appl.*, 2021, pp. 15–21.
[30] J. Guan, Y. Liu, Q. Liu, and J. Peng, "Energy-efficient amortized inference with cascaded deep classifiers," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, J. Lang, Ed., Stockholm, Sweden, 2018, pp. 2184–2190.
[31] Z. Wu et al., "BlockDrop: Dynamic inference paths in residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8817–8826.
[32] J. Xin, R. Nogueira, Y. Yu, and J. Lin, "Early exiting BERT for efficient document ranking," in *Proc. SustaiNLP, Workshop Simple Efficient Natural Lang. Process.*, 2020, pp. 83–88.
[33] D. J. Bajpai, V. K. Trivedi, S. L. Yadav, and M. K. Hanawal, "SplitEE: Early exit in deep neural networks with split computing," in *Proc. 3rd Int. Conf. AI-ML Syst.*, Bangalore, India, 2023, pp. 1–9.
[34] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, "BERT loses patience: Fast and robust inference with early exit," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 18330–18341.
[35] J. Xin, R. Tang, Y. Yu, and J. Lin, "BERxiT: Early exiting for BERT with better fine-tuning and extension to regression," in *Proc. 16th Conf. Eur. Chapter Assoc. Comput. Linguistics, Main Vol.*, 2021, pp. 91–104.
[36] H. Li, H. Zhang, X. Qi, R. Yang, and G. Huang, "Improved techniques for training adaptive deep networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1891–1900.
[37] J. Jiang, X. Wang, M. Long, and J. Wang, "Resource efficient domain adaptation," in *Proc. 28th ACM Int. Conf. Multimedia*, 2020, pp. 2220–2228.
[38] Z. Zhao, K. Wang, N. Ling, and G. Xing, "EdgeML: An automl framework for real-time deep learning on the edge," in *Proc. Int. Conf. Internet-of-Things Des. Implementation*, 2021, pp. 133–144.
[39] R. G. Pacheco, K. Bochie, M. S. Gilbert, R. S. Couto, and M. E. M. Campista, "Towards edge computing using early-exit convolutional neural networks," *Information*, vol. 12, no. 10, 2021, Art. no. 431.
[40] P. Haseena Rahmath, V. Srivastava, and K. Chaurasia, "A strategy to accelerate the inference of a complex deep neural network," in *Proc. Data Analytics Manage., ICDAM 2022*, 2023, pp. 57–68.
[41] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "DeeBERT: Dynamic early exiting for accelerating BERT inference," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, Eds., 2020, pp. 2246–2251.
[42] Y. Liu, T. Hao, H. Liu, Y. Mu, H. Weng, and F. L. Wang, "OdeBERT: One-stage deep-supervised early-exiting BERT for fast inference in user intent classification," *ACM Trans. Asian Low-Resource Lang. Inf. Process.*, vol. 22, no. 5, pp. 1–18, May 2023.
[43] Y. Shen, P. -S. Huang, J. Gao, and W. Chen, "ReasoNet: Learning to stop reading in machine comprehension," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 1047–1055.
[44] H. Fan, Z. Xu, L. Zhu, C. Yan, J. Ge, and Y. Yang, "Watching a small portion could be as good as watching all: Towards efficient video classification," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 705–711.
[45] J. Shao, H. Zhang, Y. Mao, and J. Zhang, "Branchy-GNN: A device-edge co-inference framework for efficient point cloud processing," in *Proc.2021 IEEE Int. Conf. Acoust., Speech Signal Process.*, 2021, pp. 8488–8492.
[46] L. Yang, Z. Zheng, J. Wang, S. Song, G. Huang, and F. Li, "AdaDet: An adaptive object detection system based on early-exit neural networks," *IEEE Trans. Cogn. Devlop. Syst.*, vol. 16, no. 1, pp. 332–345, Feb. 2024.
[47] A. Bojchevski et al., "Scaling graph neural networks with approximate pagerank," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 2464–2473.
[48] M. Chen et al., "Scalable graph neural networks via bidirectional propagation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 14556–14566.
[49] D. Shen, C. Qin, Q. Zhang, H. Zhu, and H. Xiong, "Handling over-smoothing and over-squashing in graph convolution with maximization operation," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 36, no. 5, pp. 8743–8756, May 2025.
[50] H. Li, C. Li, J. Zhang, Y. Ouyang, and W. Rong, "Addressing over-squashing in GNNs with graph rewiring and ordered neurons," in *Proc. 2024 Int. Joint Conf. Neural Networks*, 2024, pp. 1–8.
[51] Y. Zhuo, X. Zhou, and J. Wu, "Training graph convolutional neural network against label noise," in *Proc. 28th Int. Conf. Neural Inf. Process.*, 2021, pp. 677–689.
[52] Y. Li, J. Yin, and L. Chen, "Unified robust training for graph neural networks against label noise," in *Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining*, 2021, pp. 528–540.
[53] X. Li et al., "Contrastive learning of graphs under label noise," *Neural Networks*, vol. 172, 2024, Art. no. 106113.
[54] W. Prummel, J. H. Giraldo, A. Zakharova, and T. Bouwmans, "Inductive graph neural networks for moving object segmentation," in *Proc. 2023 IEEE Int. Conf. Image Process.*, 2023, pp. 2730–2734.
[55] X. Shen, P. Lio, L. Yang, R. Yuan, Y. Zhang, and C. Peng, "Graph rewiring and preprocessing for graph neural networks based on effective resistance," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 11, pp. 6330–6343, Nov. 2024.
[56] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Inf. Retrieval*, vol. 3, pp. 127–163, 2000.
[57] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, pp. 93–93, 2008.
[58] C. L. Giles, K. D. Bollacker, and S. Lawrence, "CiteSeer: An automatic citation indexing system," in *Proc. 3rd ACM Conf. Digit. Libraries*, 1998, pp. 89–98.
[59] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2015, pp. 43–52.
[60] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," in *Proc. Relational Representation Learn. Workshop*, 2018.
[61] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *J. Complex Networks*, vol. 9, no. 2, 2021, Art. no. cnab014.
[62] J. Tang, J. Sun, C. Wang, and Z. Yang, "Social influence analysis in large-scale networks," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2009, pp. 807–816.
[63] W. Hu et al., "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 22118–22133.
[64] S. K. Maurya, X. Liu, and T. Murata, "Feature selection: Key to enhance node classification with graph neural networks," *CAAI Trans. Intell. Technol.*, vol. 8, no. 1, pp. 14–28, 2023.
[65] Y. Luo, L. Shi, and X. -M. Wu, "Classic GNNs are strong baselines: Reassessing GNNs for node classification," in *Proc. 38th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track*, 2024.

Haseena Rahmath P received the B.Tech. degree in computer science and engineering from the Cochin University of Science and Technology, Kochi, Kerala, India, and the M.Tech. degree from Maharishi Dayanand University, Rohtak, Haryana, India. She is currently working toward the Ph.D. degree with Bennett University, Greater Noida, India. She has broad experience in the software industry, spanning software development, systems engineering, solution design, and large-scale integration. Her research interests include early-exit deep learning, interpretability, and neural network optimization.

Kuldeep Chaurasia received the M.Tech. degree in remote sensing and GIS from MANIT Bhopal, Bhopal, India, and the Ph.D. degree in geomatics from IIT Roorkee, Roorkee, India. He was with NRSC–ISRO, Hyderabad, India, contributing to national geospatial projects. He is currently an Associate Professor with Bennett University, Greater Noida, India. He has authored more than 35 publications in the field of AI and geospatial science. His research interests include machine learning for geospatial applications, hyperspectral classification, SAR/optical data analysis, flood mapping, and geo-blockchain.

Anika received the M.Eng. degree in computer science and engineering and the Ph.D. degree in adaptive learning systems from Thapar University, Patiala, India, in 2014 and 2022, respectively. She is currently an Assistant Professor with the Department of Computer Science and Engineering, School of ICT, Gautam Buddha University, Greater Noida, India. Her research interests include educational data science, learning analytics, and artificial intelligence.