MUTUAL INFORMATION AND LATENCY-AWARE ADAPTIVE CONTROL FOR RESOURCE-EFFICIENT GRAPH NEURAL NETWORKS

YAN-FEI MA[1], DAO-ZHENG QU[2]

[1]Department of Computer Science, Fairleigh Dickinson University, Vancouver, V6B 2P6, Canada
[2]Department of Computer Science, University of Liverpool, Liverpool, L69 3DR, UK
E-MAIL: yanfei.ma@ieee.org, daozheng.qu@ieee.org

Abstract:

Despite their many potential applications, graph neural networks (GNNs) are challenging to deploy on low-resource devices like Internet of Things nodes and mobile platforms due to their high energy and processing demands. We propose MLC-GNN, a lightweight architecture that dynamically optimizes depth, precision, and channel width during inference. MLC-GNN uses a middle-layer controller based on real-time mutual information estimate and a PID-based energy guard to offer efficient adaptation under strict energy and latency constraints. Extensive experiments on four benchmark datasets and five baselines demonstrate that MLC-GNN achieves competitive or superior accuracy and precision, while reducing energy consumption and inference delay by up to 50%. These results show the utility of MLC-GNN in enabling energy-efficient GNN inference on edge devices.

Keywords:

Graph Neural Networks, Resource-Constrained Inference, Energy-Efficient Deep Learning, Dynamic Model Adaptation, Mutual Information Estimation, Quantized GNNs

1. Introduction

Graph neural networks (GNNs) are widely used for learning from graph-structured data in applications such as fraud detection, drug discovery, and traffic forecasting [?, ?], due to their ability to model non-Euclidean relationships. However, GNNs remain difficult to deploy on energy-constrained platforms such as mobile devices and IoT edge systems.

Unlike cloud systems with ample resources, edge devices operate under tight energy and latency budgets. Standard GNNs' memory overhead, message passing, and full-precision computation are too costly in such settings. As a result, GNN designs that can adaptably function under different hardware budgets without sacrificing performance are becoming more and more in demand.

Despite recent efforts, most existing GNN models suffer from three critical limitations that hinder their real-world deployability: (i) Static architectures: Traditional GNNs use fixed width and depth, which prevents graceful scaling under energy constraints and leads to inefficient computation. (ii) Fixed-precision computation: Many GNNs rely on static quantization (e.g., INT8 or FP32), missing fine-grained trade-offs between accuracy and energy, and thus may overconsume resources. (iii) Unaware of information degradation:Deep GNNs suffer from cross-layer information loss, yet most models lack runtime tools to monitor or control mutual information, risking under- or overfitting under constraints.

Together, these constraints limit GNNs' ability to adjust to dynamic execution contexts, especially in applications that are energy-conscious and latency-sensitive [?, ?]. A new class of designs that can both intelligently react to runtime signals and compress computation is needed to address them.

In this paper, we offer MLC-GNN (Mutual Information and Latency-aware Control for GNNs), a lightweight GNN framework built for energy-constrained inference. MLC-GNN has a middle-layer controller that dynamically modifies the channel width, quantization precision, and message transmission depth by monitoring real-time mutual information and device energy feedback. To enable robust inference on edge devices, this adaptive control is further controlled by a PID-based energy guard that maintains cumulative energy consumption within goal bounds.

Our contributions are summarized as follows:

- We design MLC-GNN, the GNN architecture that dynamically controls depth, precision, and channel width based on real-time mutual information and device energy feedback.
- We introduce a bound-aware control law and a PID-based energy guard to ensure stable, energy-efficient inference under resource constraints.
- Experiments on four datasets show up to 50% energy savings and improved precision over five baselines.

2. Related Work

Static GNN Architectures. GraphSAGE [?] and GCNs [?] are widely used for semi-supervised classification. Their static design—fixed depth, width, and precision—limits adaptability. Even GraphSAGE's sampling lacks runtime flexibility. Fixed-depth models underperform on graphs with high resistance [?]. S-GCN [?] reduces latency via simplification but remains rigid.

Quantized and Compressed GNNs. QGNN [?], TinyGNN [?], and GNNPack [?] use low-bit ops and system-level optimizations. However, fixed settings limit adaptability. Without feedback, quantized models may over-prune and generalize poorly.

Energy- and Latency-Aware Deep Learning. SkipNet [?], DynamicNet [?], and CondConv [?] use input-aware routing. But in GNNs, neighborhood aggregation and graph irregularity make dynamic computation harder. DeltaGNN [?] and HGNAS [?] address this but rely on costly meta-learning.

Mutual Information Estimation in GNNs. MI degrades as GNNs deepen [?]. InfoMax GNN [?] preserves MI during training but does not affect inference. MLC-GNN uses MI as a runtime signal to adaptively tune GNN architecture based on data and energy constraints.

3. Methodology

We present MLC-GNN, a modular and lightweight framework for GNN inference that uses less energy. MLC-GNN dynamically modifies three fundamental architectural dimensions—depth, quantization precision, and channel width—during runtime, in contrast to conventional GNNs that use static topologies and uniform precision. Real-time energy status and mutual information (MI) feedback are used to adaptively make these changes.

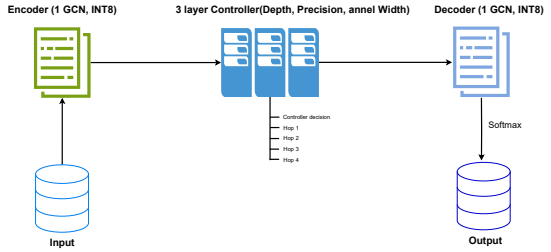

FIGURE 1. Overview of the MLC-GNN architecture. The input features are first embedded by a low-precision encoder (1-layer GCN, INT8). The 3-layer controller then observes real-time mutual information $\hat{\mathcal{I}}^{\ell}$ and current energy state E_h, and outputs decisions to dynamically adjust depth (L), precision (κ), and channel width (C). These settings configure the message-passing MLC blocks (not explicitly shown), followed by an INT8 decoder GCN and softmax classification. A PID-based energy guard modulates controller decisions by injecting feedback from recent energy deviation signals.

3.1. System Overview

As illustrated in Figure 1, the MLC-GNN architecture consists of five key components: (1) Encoder: A fixed single-layer GCN that encodes input features into a low-precision latent space (INT8, 128-dim); (2) Middle-layer Controller: A lightweight neural controller that observes mutual information $\hat{\mathcal{I}}^{\ell}$ and cumulative energy state, then issues actions for upcoming hops; (3) MLC-Block: A configurable GCN layer supporting dynamic hop count L, precision κ, and channel width C; (4) Decoder: A single-layer INT8 GCN for mapping features to class logits; (5) PID Energy Guard: A PID controller maintaining energy usage within bounds using a sliding window of inference traces.

Each inference pass follows the pipeline: Input $\rightarrow$ Encoder $\rightarrow$ Controller $\rightarrow$ MLC-Blocks $\rightarrow$ Decoder $\rightarrow$ Output. The controller dynamically adjusts the runtime configuration based on observed mutual information $\hat{\mathcal{I}}^{\ell}$ and current energy state E_h, selecting the number of message-passing layers (depth L), bit precision (κ), and active channels (C). These decisions are further modulated by a PID-based energy guard to ensure energy remains within the target envelope.

To better illustrate the overall structure and execution process of MLC-GNN, we summarize the complete inference and training workflow in Algorithm 1.

Algorithm 1 MLC-GNN: Adaptive Inference and Training Procedure

Input: Graph $G = (V, E, \mathbf{X})$, energy budget $E_h^{\max}$
Output: Node classification results $\{\hat{y}_v\}_{v \in V}$
Initialization:
  Initialize encoder $\mathcal{E}$, decoder $\mathcal{D}$, controller $\mathcal{C}$, PID guard
  Set controller state: $\kappa \leftarrow 8$, $C \leftarrow 128$
Warm-up Phase (30 epochs):
for epoch = 1 to 30 do
    Forward pass with static config: $(L = 4, \kappa = 8, C = 128)$
    Update model via $\mathcal{L}_{\text{CE}}$
end for
Adaptive Phase (100 epochs):
for epoch = 31 to 130 do
  for each node $v \in V$ do
    Encode:
      $\mathbf{h}_v^0 \leftarrow \mathcal{E}(\mathbf{x}_v)$
    Initialize: $E_h \leftarrow 0$
    for $\ell = 1$ to $L_{\max}$ do
      Estimate Mutual Information: $\hat{\mathcal{I}}^\ell \leftarrow \text{MINE}(\mathbf{h}_v^{\ell-1})$
      Compute Bound: $\mathcal{B}_\ell \leftarrow \exp(-\alpha \ell R_{\text{avg}}) \cdot \exp(-\beta(E_h^{\max} - E_h)) \cdot 2^{-\kappa}$
      Controller Decision:
        $g_\ell \leftarrow \hat{\mathcal{I}}^\ell - \mathcal{B}_\ell$
      if $g_\ell < \tau_{\text{skip}}$ then
          Skip hop ℓ
          continue
      else if $g_\ell > \tau_{\text{boost}}$ and $E_h < E_h^{\max}$ then
          Increase κ or add hop
      end if
        Update $C \leftarrow \text{clip}(C + \eta g_\ell, 64, 256)$
      Forward Hop: $\mathbf{h}_v^\ell \leftarrow \text{MLC-Block}(\mathbf{h}_v^{\ell-1}, \kappa, C)$
        Accumulate energy E_h
    end for
    Decode: $\hat{y}_v \leftarrow \mathcal{D}(\mathbf{h}_v^L)$
  end for
  Update: Train with loss $\mathcal{L} = \text{CE} + \lambda \cdot \text{Energy}$
  PID Update: Adjust Δ_{pid} using $e_t = E_h - E_{\text{target}}$
end for
Fine-tuning Phase (50 epochs):
for epoch = 131 to 180 do
  Freeze controller; refine weights using CE loss
end for
return $\{\hat{y}_v\}_{v \in V}$

3.2. MLC-Block Design

The MLC-Block forms the core of adaptive inference. At hop ℓ, it performs a GCN-style neighborhood aggregation, followed by:

(1)Layer normalization; (2)Non-linear activation (ReLU or Swish); (3)Bit-width specific quantization (e.g., 4/8/16-bit fixed-point); (4)Optional gating mechanism to bypass layer computation when necessary.

Runtime projection modules choose subspaces of size $C \in \{64, 128, 192, 256\}$, and all weight matrices are pre-initialized at 256 channels to guarantee compatibility with runtime switching. By turning on several quantization layers, the quantization precision is dynamically chosen. Rapid architectural reconfiguration is made possible by this modular design, which eliminates the need for model reinitialization or retraining.

3.3. Controller Architecture and Algorithm

The controller is a three-layer MLP ($\dim_{\text{input}} = 3$, $\dim_{\text{hidden}} = 64$, $\dim_{\text{output}} = 3$) that takes as input the estimated mutual information $\hat{\mathcal{I}}^\ell$, the cumulative energy E_h, and the average effective resistance R_{avg}.

It determines whether to skip the next hop, increase precision κ, and update channel width C, based on the gap between observed MI and its theoretical bound.

3.4. Bound-Aware MI Control Law

To avoid overfitting and over-computation, we propose a control signal $\mathcal{B}_\ell$ that acts as a soft upper bound on required MI. It decays exponentially with both hop depth and residual energy margin:

$$\mathcal{B}_\ell = \exp(-\alpha \ell R_{\text{avg}}) \cdot \exp(-\beta(E_h^{\max} - E_h)) \cdot 2^{-\kappa} \quad (1)$$

where α, β are tunable decay parameters. The MI estimation $\hat{\mathcal{I}}^\ell$ is computed using a lightweight MINE probe with 512 hidden units and a shared discriminator across hops, amortizing cost.

3.5. PID-Based Energy Guard

The energy guard monitors consumption over a sliding window of 128 inferences. It employs PID control to modulate aggressiveness of controller decisions. Let E_{target}

denote the desired per-inference energy. At each step, we compute:

$$\Delta_{\text{pid}} = P \cdot e_t + I \cdot \sum_{i=1}^{t} e_i + D \cdot (e_t - e_{t-1}) \tag{2}$$

where $e_t = E_t - E_{\text{target}}$ is the error signal. The value Δ_{pid} is injected into the controller input to act as a correction bias.

3.6. Training Schedule and Stability Measures

To ensure smooth convergence of the adaptive model, training proceeds in three phases. Warm-up Phase (30 epochs): The model uses a static configuration ($L = 4$, $\kappa = 8$, $C = 128$), and the controller remains inactive. Adaptive Phase (100 epochs): The controller is activated, and the model is optimized using the composite loss:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \cdot \mathcal{L}_{\text{energy}}, \quad \lambda = 0.1 \tag{3}$$

where $\mathcal{L}_{\text{energy}}$ denotes the average per-inference energy consumption estimated during forward passes. Fine-tuning Phase (50 epochs): The controller is frozen, and model weights are refined.

Throughout, we use 5×10^{-4} for weight decay and cosine learning rate decay. We employ EMA smoothing with a decay factor of 0.95 and batch-wise gradient clipping to stabilize MI estimation.

4. Evaluation

The effectiveness and practical deployability of the proposed MLC-GNN model on low-power edge devices are evaluated under resource-constrained scenarios. Verifying if its dynamic adjustment of depth, accuracy, and channel width can sustain predictive performance while drastically lowering energy consumption and inference delay is the fundamental objective.

4.1. Experimental Setup

Datasets. Experiments are conducted on four widely-used graph datasets. Cora and Citeseer [?] are small citation networks with low node degrees. Amazon Photo [?] is a medium-sized co-purchase graph with denser connections. ogbn-arxiv [?] is a large-scale citation graph from the Open Graph Benchmark with over 169k nodes. This selection allows evaluation across diverse graph structures.

Setup. All models are deployed on Raspberry Pi 5 and NVIDIA Jetson Nano. We use PyTorch 2.0 + CUDA 11.8, apply quantization-aware training (QAT) with static INT8 calibration, and record energy using on-device sensors (TI INA219 on Pi; Jetson built-in monitor) with 1-second intervals. Batch size is 128; all results are averaged over 5 runs.

Metrics. We evaluate test set accuracy, precision (true positives over predicted positives), energy consumption (mJ per inference), and latency (ms per inference).

Baselines. We compare MLC-GNN against five models: GCN [?], a two-layer FP32 GCN; GraphSAGE [?], a scalable aggregator-based GNN; QGNN [?], an 8-bit quantized GCN; S-GCN [?], a simplified GCN with linear propagation; and DeltaGNN [?], which adaptively skips message-passing layers. All models are retrained using the same QAT pipeline and evaluated under identical hardware conditions.

4.2. Main Results and Analysis

Table 1 summarizes the performance of MLC-GNN and baseline models across four benchmark datasets. While GraphSAGE achieves the highest accuracy in most cases, MLC-GNN consistently delivers the best energy and latency performance. It also maintains competitive accuracy and often yields the highest precision, demonstrating its effectiveness under resource-constrained scenarios.

TABLE 1. Performance comparison across datasets and models.

Dataset	Model	Accuracy (%)	Precision (%)	Energy (mJ)	Latency (ms)
Cora	GCN	82.1	81.7	58	5.1
	GraphSAGE	83.0	82.6	61	5.4
	QGNN	80.6	79.5	36	4.3
	S-GCN	81.0	80.8	32	4.3
	DeltaGNN	82.3	81.9	30	4.2
	MLC-GNN	82.7	83.1	28	4.2
Citeseer	GCN	71.9	71.5	62	5.7
	GraphSAGE	72.8	72.3	65	5.9
	QGNN	69.0	67.8	39	4.5
	S-GCN	70.6	70.1	35	4.4
	DeltaGNN	71.5	71.0	31	4.4
	MLC-GNN	71.2	71.6	30	4.3
Amazon Photo	GCN	89.4	89.0	97	9.5
	GraphSAGE	90.1	89.3	102	9.9
	QGNN	86.8	85.9	56	7.3
	S-GCN	88.2	87.5	51	7.1
	DeltaGNN	88.9	88.6	49	7.1
	MLC-GNN	89.2	89.5	48	7.0
ogbn-arxiv	GCN	71.2	70.8	152	16.8
	GraphSAGE	72.3	72.0	160	17.2
	QGNN	67.5	66.3	89	12.5
	S-GCN	69.8	69.0	81	12.0
	DeltaGNN	70.4	70.2	78	11.9
	MLC-GNN	71.0	71.4	76	11.9

In particular, on Cora, MLC-GNN reduces energy consumption by 52% and latency by 18% compared to GCN,

while achieving the highest precision and competitive accuracy (within 0.3% of GraphSAGE). On Citeseer, MLC-GNN achieves a precision score close to GraphSAGE (71.6% vs 72.3%), with the lowest energy and latency—cutting energy usage by more than 50%. On Amazon Photo, MLC-GNN offers a highly favorable trade-off between accuracy and energy, achieving near-top accuracy at significantly lower power cost, as shown in Figure 2. On ogbn-arxiv, MLC-GNN maintains strong predictive performance (1.3% lower accuracy than GraphSAGE), while reducing latency by approximately 30% and energy consumption by over 50%.

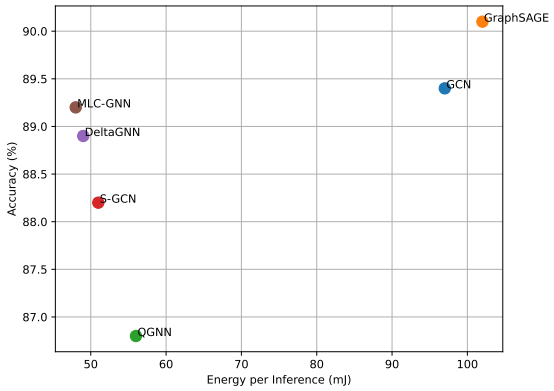

FIGURE 2. The energy-accuracy trade-off for all methods on Amazon Photo

In summary, MLC-GNN strikes a strong balance between performance and efficiency: it consistently reduces energy and latency by over 50% with only marginal accuracy loss, and achieves the highest precision on most datasets. This confirms its suitability for low-resource GNN inference.

4.3. Ablation Study

To isolate the effect of each adaptive module, we conduct an ablation study on Amazon Photo:

TABLE 2. Ablation results on Amazon Photo (Accuracy / Precision / Energy / Latency)

Variant	Accuracy (%)	Precision (%)	Energy (mJ)	Latency (ms)
Full MLC-GNN	89.2	89.5	48	7.0
w/o MI control	88.1	87.7	66	8.5
w/o PID guard	88.6	88.3	58	7.9
Fixed C (128)	88.9	88.6	54	7.5

Each ablation variant disables a key adaptive component from the full model: w/o MI control disables mutual information estimation and hop-skipping logic; w/o PID guard removes energy-aware PID correction from the controller input; and Fixed C sets channel width $C = 128$ statically without dynamic adaptation.

This shows that: (1)Disabling mutual information estimation causes the largest energy increase (+37.5%) and significantly reduces precision (–1.8%); (2)Removing PID control leads to unstable consumption and minor degradation in precision; (3)Static channel width underutilizes resource flexibility, resulting in moderate energy loss and precision drop.

4.4. Sensitivity Analysis

We examine MLC-GNN's sensitivity to important control thresholds and architectural elements in order to respond to Q3 from our evaluation scope. In particular, we change the starting channel width C, the skip threshold τ_{skip}, and the boost threshold τ_{boost} and see how these affect the accuracy and energy efficiency of the model.

We explore how MLC-GNN responds to different values of the controller's thresholds $\tau_{\text{skip}}, \tau_{\text{boost}}$ and channel width C.

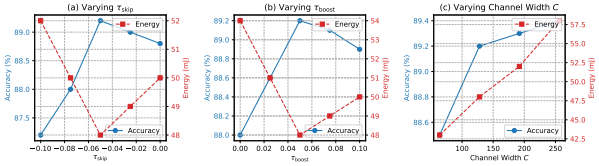

FIGURE 3. Sensitivity of energy and accuracy to controller parameters.

We observe that: - τ_{skip} too low (< -0.05) leads to excessive layer skipping and accuracy drop; - Increasing τ_{boost} reduces over-computation; - Dynamic C helps maintain efficiency under fluctuating energy budgets.

4.5. Evaluation Summary

MLC-GNN exhibits strong generalization and energy efficiency on all graphs and devices that have been tested. Its primary benefit is its runtime flexibility, which allows it to selectively grow or shrink in response to power availability and graph complexity. MLC-GNN dynamically adapts its compute budget to real-world restrictions, which makes it more appropriate for edge deployments than QGNN or

GraphSAGE, which use predefined techniques. Compared to lightweight baselines such as S-GCN and DeltaGNN, MLC-GNN offers competitive accuracy and significantly higher precision while retaining its energy advantages.

5. Conclusion

In this work, we present MLC-GNN, a novel graph neural network architecture for edge devices that are subject to energy constraints. MLC-GNN provides fine-grained control over the energy-to-latency-to-accuracy trade-off by dynamically modifying the model's depth, precision, and channel width in response to real-time mutual information estimates and energy feedback. A PID-based energy guard and a lightweight middle-layer controller, both of which have low overhead requirements, make this design possible. Comprehensive tests on four graph benchmarks show that MLC-GNN matches or exceeds baseline accuracy and precision while cutting energy and latency by up to 50%. These findings demonstrate how well MLC-GNN adapts to various runtime limitations, which makes it a good fit for deployment on real-world platforms including embedded AI accelerators, mobile CPUs, and Internet of Things nodes.

Looking forward, we identify two promising directions for extending this work: Dynamic neighbor selection, which integrates topology-aware routing policies that adapt the neighborhood aggregation pattern per node or per hop to further reduce redundant computation; and Extreme compression, which explores ultra-low-bit quantization (e.g., binary/ternary GNNs) and structured pruning to push the boundaries of model compactness without compromising inference robustness.

Overall, MLC-GNN represents a step toward making GNN inference both adaptive and resource-aware, paving the way for truly deployable graph intelligence at the edge.