ONLINE UNIFIED GRAPH PRUNING FOR COMPACT GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) have achieved strong performance on graph learning tasks, but their deployment is often limited by the computational cost caused by dense graph structures and high-dimensional hidden representations. Existing pruning methods usually focus on either graph sparsification or model compression, and often treat these two pruning dimensions independently. In this paper, we study **O**nline **U**nified **G**raph **P**runing (OUGP), which jointly learns graph-edge sparsity and parameter sparsity during GNN training. The key idea is to use hidden representations as feedback signals to couple graph pruning and parameter pruning, so that the compact model can preserve task-relevant structural and semantic information. To improve deployment robustness under distribution shift, we further introduce an **E**nergy-**S**hift **P**arameter-**M**ask adaptation module (ESPM), which adjusts parameter selection according to unlabeled target-domain statistics while preserving the compact structure. Experiments compare OUGP+ESPM with dense and existing unified pruning baselines, and evaluate its accuracy, sparsity, efficiency, and robustness under distribution shift. For instance, under 30% graph and 30% model sparsity, OUGP+ESPM improves accuracy by 0.92%↑ on Cora, while reducing training FLOPs by 25%↓ compared with the dense GCN. The code is anonymously available at link.

1 INTRODUCTION

Graph neural networks (GNNs) have emerged as a powerful framework for learning on graph-structured data, leveraging both node attributes and structural dependencies among entities (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018; Wu et al., 2019). This makes them useful for node classification, link prediction, graph representation learning, and relational inference in citation and social networks (Sen et al., 2008; Kipf & Welling, 2017; Hamilton et al., 2017). Their scope has expanded across various fields, serving as a flexible computational interface for applications where interaction structure is as important as input features and relational patterns.

However, this expressive relational computation introduces substantial deployment challenges. A GNN layer couples neighborhood aggregation over the graph with feature transformation through model parameters. As graphs continue to grow, the resulting computation can incur significant memory and computational overhead, with costs scaling with graph size, hidden dimensions, and network depth (Chiang et al., 2019; Zeng et al., 2020; Hu et al., 2020; Fey et al., 2021). This motivates pruning to allocate a limited computational budget to the graph structures and model parameters that best preserve the learned representations.

We argue that pruning should therefore be viewed as **representation-conditioned support allocation**, rather than static removal. Because graph aggregation and feature transformation jointly determine the hidden representation, the importance of an edge depends on the retained parameter support, and vice versa. Thus, graph and parameter pruning are inherently coupled, and independently optimizing their supports can lead to suboptimal decisions. Moreover, as hidden representations evolve during training, the optimal sparse support can change across layers and optimization stages. Our analysis reveals that pruning-induced perturbations are strongly **layer-dependent**, while graph and parameter pruning exhibit **non-additive interactions** in the hidden space. This motivates the following question: ***I****) How can evolving hidden representations guide* ***layer-wise*** *and* ***interaction-aware*** *joint pruning to preserve informative graph structures and parameter supports?*

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

Existing pruning methods have attempted to identify informative graph structures or model parameters under constrained budgets. Structure-driven approaches select graph supports by exploiting topology or connectivity patterns, while parameter-driven approaches remove weights or channels based on predefined importance criteria (Rong et al., 2020; Zheng et al., 2020; Li et al., 2017). Recent mask- or score-driven approaches further learn pruning decisions during training to obtain compact models (Chen et al., 2021; Liu et al., 2024; Liao et al., 2025). However, these approaches mainly evaluate the importance of individual structures or parameters, without explicitly modeling which sparse support best preserves the evolving hidden representations after pruning.

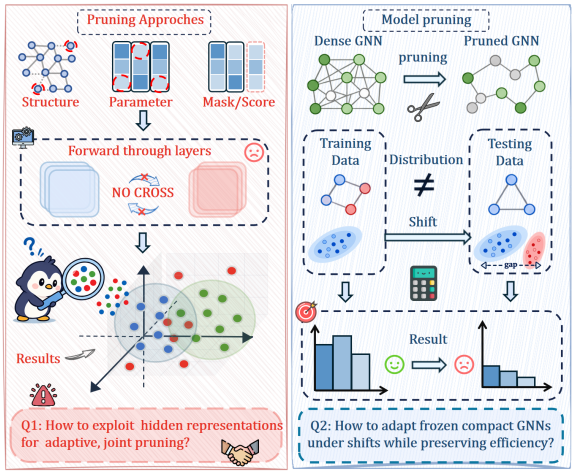


Figure 1: **Problem Illustration.**We describe the challenges in robust GNN pruning: **I)** Existing pruning methods independently optimize graph structures, parameters or masks, ignoring their coupled effects on hidden representations. This leads to suboptimal pruning decisions and representation degradation. **II)** Pruned models struggle under distribution shift, degrading prediction performance.

The compact model also faces a deployment challenge: the target distribution may differ from the source training distribution. Such **distribution shifts** can change the hidden representations and the parameter supports that are useful for prediction. Existing test-time adaptation methods often rely on normalization updates, entropy minimization, or iterative parameter optimization (Sun et al., 2020; Wang et al., 2021; Niu et al., 2022; 2023), which may introduce additional computation or modify the deployed model. This motivates a lightweight adaptation mechanism that adjusts parameter support under deployment shifts while preserving the compact budget: ***II)*** *How can we improve the* ***adaptability*** *of compact GNNs under* ***distribution shifts*** *while maintaining their efficiency and structural integrity?*

To address ***I***, we develop a source-to-deployment framework centered on **O**nline **U**nified **G**raph and **P**arameter Pruning (**OUGP**). OUGP jointly optimizes graph-edge pruning and parameter sparsification under explicit budgets. Its **L**ayer-wise **H**idden **C**oupling **U**tility (**LHCU**) derives pruning utilities from **hidden-state perturbations** and **graph–parameter interactions**, enabling representation-conditioned support allocation during optimization. The learned graph and parameter scores are finally hardened to obtain a compact GNN with sparse graph and parameter supports.

To tackle ***II***, we further introduce an **E**nergy-**S**hifted **P**arameter-**M**ask test-time adaptation (ESPM) module. Starting from the compact model after source training, ESPM estimates unlabeled target-side **energy shifts** and propagates them to parameter entries through the pretrained parameter structure. The source parameter scores provide a stable ranking reference, while **energy-guided corrections re-rank** parameter importance and construct a target-specific parameter mask under the pruning budget. At inference, the adapted compact model performs a forward-only pass with the selected parameter support. Together, these components establish a source-to-deployment framework for efficient and adaptable compact GNNs. Our principal contributions are summarized as follows:

❶ ***Hidden-State-Guided Coupled Joint Pruning.*** We introduce LHCU to derive pruning utilities from hidden-state perturbations induced by different pruning supports. By comparing the resulting hidden representations, LHCU quantifies the effects of graph and parameter pruning on representation preservation and captures their non-additive interactions. Based on these utilities, OUGP jointly optimizes graph-edge and parameter pruning layer by layer under separately controlled budgets.
❷ ***Memory-Augmented Pruning Optimization.*** We incorporate a low-rank memory MLP to accumulate pruning information across layers and training epochs, enabling the pruning strategy to learn from historical optimization states rather than making independent decisions at each layer or epoch.
❸ ***Energy-Shifted Parameter-Mask Adaptation.*** We introduce ESPM, a forward-only test-time adaptation strategy that estimates channel-wise energy shifts from unlabeled target features and uses them to re-rank source parameter scores, yielding a target-specific parameter mask under the original budget, improving robustness under distribution shifts.

2 Related Work

2.1 Graph sparsification for GNNs

Graph sparsification aims to remove redundant or noisy edges from the input graph(Rong et al., 2020; Zheng et al., 2020) while preserving task-relevant structural information (Rong et al., 2020; Zheng et al., 2020; Zeng et al., 2020; Chiang et al., 2019). For GNNs, this can reduce message-passing cost and sometimes improve robustness by suppressing harmful connections (Rong et al., 2020; Zeng et al., 2020). Typical graph pruning strategies select edges according to structural heuristics, learned importance scores, or training-time sensitivity (Li et al., 2020; 2021; Liu et al., 2023; Kosman et al., 2022; Rong et al., 2020; Zheng et al., 2020). However, graph-only pruning does not reduce the hidden representation dimension, so the feature transformation cost can remain high.

2.2 Model and channel pruning

Model pruning removes unnecessary model parameters or hidden dimensions to obtain compact neural networks, including magnitude-based pruning, channel pruning, and sparse subnetworks discovery (Li et al., 2017; He et al., 2017; Liu et al., 2017; Lee et al., 2019; Wang et al., 2020; Frankle & Carbin, 2019; Sanh et al., 2020). In GNNs, hidden-channel pruning directly reduces the width of hidden representations and has been explored as an effective strategy for reducing parameter and computation overhead in graph models (Chen et al., 2021; Liu et al., 2024). This can lower memory usage and computation in linear transformations. However, channel-only pruning keeps the original graph structure unchanged, so message passing may still involve redundant edges.

2.3 Unified pruning for graph neural networks

Unified pruning methods attempt to jointly optimize multiple components of a GNN, including graph edges, model parameters, and hidden channels (Chen et al., 2021; Liu et al., 2024; Jiang et al., 2022; Wang et al., 2023; Liao et al., 2025). These methods are closer to the goal of compact GNN deployment by jointly reducing structural and parameter redundancy, following the trend of scalable and efficient graph learning (Chen et al., 2021; Liu et al., 2024; Fey et al., 2021).However, existing approaches commonly rely on learned or predefined sparsification mechanisms, separate importance estimation, or staged pruning procedures (Chen et al., 2021; Liu et al., 2024; Wang et al., 2023; Liao et al., 2025). As a result, they may not fully capture the dynamic interaction between graph structure and hidden representations during training, leaving the potential of representation-aware pruning underexplored for constructing more adaptive and efficient compact GNNs.

2.4 Test-time robustness of compact models

Pruned models can be more sensitive to test-time distribution shift because pruning removes part of the model capacity. When test data differ from training data, model predictions can become less reliable, motivating test-time adaptation under distribution shifts (Sun et al., 2020; Wang et al., 2021; Niu et al., 2022; 2023; Liang et al., 2020; Zhang et al., 2022). This motivates evaluating pruning methods not only on clean test data, but also under perturbed or shifted test conditions.However, most existing test-time adaptation approaches are developed for dense models, while the robustness of compact GNNs after pruning remains less explored.

3 Preliminary

3.1 Preliminary

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote an input graph with node features $X \in \mathbb{R}^{N \times F}$ and adjacency matrix $A \in \{0,1\}^{N \times N}$. For node classification, a GNN produces node logits $Z = f_\theta(A, X)$, where θ denotes the trainable backbone parameters. At layer l, the hidden representation is denoted by $H^{(l)} \in \mathbb{R}^{N \times d_l}$, where each column corresponds to a hidden channel.

3.2 Problem Formulation

OUGP considers a graph-edge mask $M_g \in \{0,1\}^{|\mathcal{E}|}$ and a parameter mask $M_p \in \{0,1\}^{|\theta|}$. The corresponding masked graph and model parameters are written as $\widetilde{A} = \mathrm{Mask}(A, M_g)$ and $\widetilde{\theta} =$

$\theta \odot M_p$. Given graph and parameter sparsity budgets $\rho_g^\star$ and $\rho_p^\star$, the source training objective is $\min_{\theta, M_g, M_p} \mathcal{L}_{\text{src}}\big(f_{\tilde{\theta}}(\widetilde{A}, X)\big)$, subject to $\rho_g = \rho_g^\star$ and $\rho_p = \rho_p^\star$.

3.3 Problem Visualization

Graph pruning and parameter pruning both change the hidden representation. Let H_D, H_G, H_C, and H_{GC} denote the hidden states under dense, graph-only, channel-only, and joint masking, respectively. Their non-additive interaction is measured by

$$\mathcal{I}_H = H_{GC} - H_G - H_C + H_D. \quad (1)$$

Here, the joint effect may be complementary, overlapping, or redundant, and is not assumed to be a simple sum of the two individual effects. After pruning, the compact GNN is fixed for deployment. Let $\mathcal{A}_{\text{clean}}(\rho)$ and $\mathcal{A}_{\text{shift}}(\rho)$ denote clean and shifted test accuracy at sparsity level ρ. We define the direct-deployment gap as

$$\Delta_{\text{shift}}(\rho) = \mathcal{A}_{\text{clean}}(\rho) - \mathcal{A}_{\text{shift}}(\rho). \quad (2)$$

We visualize the deployment challenge of compact GNNs from two aspects. First, we examine whether stronger pruning increases the gap between clean and shifted test performance. Second, we illustrate how test-time adaptation corrects shifted hidden representations without changing the deployed compact model or introducing additional structural modifications during inference.

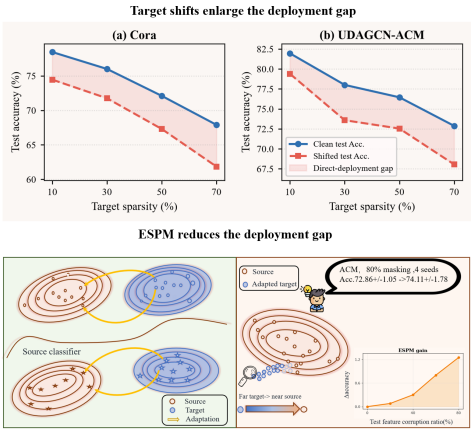


Figure 2: Problem visualization of deployment challenges for compact GNNs. The upper panel shows the pruning-induced accuracy gap under feature shifts, while the lower panel illustrates energy-gated adaptation for robust deployment without modifying the compact model.

4 Methodology

4.1 Framework Overview

Compression Motivation. Graph compression involves two forms of redundancy: graph structural redundancy in the edge set and weight redundancy in the feature transformations. Existing GNN compression methods typically optimize these dimensions separately (Zheng et al., 2020; Chen et al., 2021; Liu et al., 2024; Liao et al., 2025), leaving their interaction through hidden representations underexplored.

Two-Stage Framework. Given a source graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with features $X \in \mathbb{R}^{|\mathcal{V}| \times d_0}$ and labels Y, we use a 2-layer GCN (Kipf & Welling, 2017) with hidden state $H \in \mathbb{R}^{|\mathcal{V}| \times d}$. OUGP jointly learns graph-edge and channel policies under explicit budgets, aggregates channel scores to weight-entry scores, and hardens the resulting graph and parameter supports. The hidden width is fixed at $d = 32$. Deployment then applies energy-shifted parameter-mask adaptation to the frozen compact GCN, re-ranking parameter scores from unlabeled input-energy statistics under the same budget.

Frozen Deployment. After source training, the GNN parameters and the learned graph mask are frozen to construct the compact deployment model. Source energy statistics and parameter scores are cached, and target input energy is used to generate the deployment-time parameter mask through ESPM before the compact forward pass.

4.2 Hidden-State Utility Guided Unified Graph-Channel Pruning

4.2.1 Hidden-State Utility Construction

Four-Way Hidden-State Evaluation. We define Layer-wise Hidden Coupling Utility (LHCU) by evaluating four configurations at each hidden layer: H_{D} for dense graph and channels, H_{G} for graph-only mask, H_{C} for channel-only mask, and H_{F} for joint graph-and-channel mask.

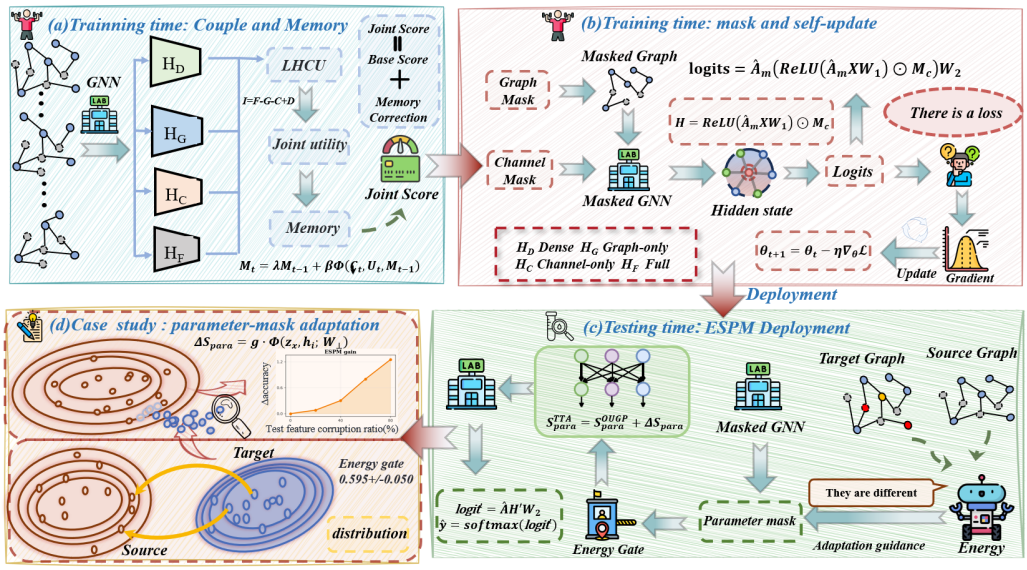


Figure 3: Overview of OUGP and the proposed test-time parameter-mask adaptation. OUGP jointly performs graph-edge and hidden-channel policy during training, materializes a frozen compact GNN, and the deployment stage re-ranks parameter scores from an unlabeled channel-wise energy shift before the target forward pass.

Perturbation Decomposition. Here, H_{D}, H_{G}, H_{C}, and H_{F} denote the corresponding hidden activations, with the layer index omitted for readability. We define the graph and channel perturbations relative to the dense representation as $\Delta H_G = H_{\mathrm{G}} - H_{\mathrm{D}}$, $\Delta H_C = H_{\mathrm{C}} - H_{\mathrm{D}}$, and define the non-additive graph–channel interaction byive graph–channel interaction by $\mathcal{I} = H_{\mathrm{F}} - H_{\mathrm{G}} - H_{\mathrm{C}} + H_{\mathrm{D}}$. Thus, ΔH_G measures the hidden-state effect of graph sparsification, ΔH_C measures the effect of hidden-channel sparsification, and $\mathcal{I}$ captures effects that cannot be explained by adding the two independent perturbations.

Interaction-Aware Utilities. For each hidden layer l and shared channel index j, the corresponding channel and edge utilities combine channel damage, graph–channel relation, and joint interaction:

$$u^C_{l,j} = \lambda_C \frac{1}{|\mathcal{V}|} \sum_i |\Delta H_{C,ij}| + \lambda_R \frac{1}{|\mathcal{V}|} \sum_i |\Delta H_{G,ij} \Delta H_{C,ij}| + \lambda_I \frac{1}{|\mathcal{V}|} \sum_i |\mathcal{I}_{ij}| \,. \tag{3}$$

$$u^V_i = \lambda_G \left\|\Delta H_G(i,:)\right\|_2 + \lambda_R \left\|\Delta H_G(i,:) \odot \Delta H_C(i,:)\right\|_2 + \lambda_I \left\|\mathcal{I}(i,:)\right\|_2 , \qquad u^G_e = \frac{u^V_{s(e)} + u^V_{t(e)}}{2}, \tag{4}$$

Layer-Wise Aggregation. With $s(e)$ and $t(e)$ denoting the endpoints of edge e, the layer-wise utilities are normalized and aggregated using weights determined by the hidden-state perturbation magnitude, relation, and interaction ratio. In particular, with

$$r_l = \frac{\|\mathcal{I}_l\|_F}{\|\Delta H_{G,l}\|_F + \|\Delta H_{C,l}\|_F + \epsilon}, \qquad \eta_l = \lambda_L \left\|\Delta H_{C,l}\right\|_F + \lambda_I r_l + \lambda_R \left|\cos(\Delta H_{G,l}, \Delta H_{C,l})\right| , \tag{5}$$

we use $\omega_l = \mathrm{softmax}_l(\eta_l)$ and obtain $u^C = \sum_l \omega_l \,\mathrm{Norm}(u^C_l)$ and $u^G = \sum_l \omega_l \,\mathrm{Norm}(u^G_l)$. These representation-level utilities are then combined with gradient utility before being written to the graph and channel memories for updating the subsequent pruning policies.es and hidden representations.

Gradient–Representation Fusion. For graph and channel logits ℓ^G and ℓ^C, respectively, the gradient utilities are $g^G = |\nabla_{\ell^G} \mathcal{L}_{\mathrm{src}} \odot \boldsymbol{G}|$, $g^C = |\nabla_{\ell^C} \mathcal{L}_{\mathrm{src}} \odot \boldsymbol{C}|$. The resulting utilities are mixed with the normalized gradient signals before they are written to the graph and channel memories:

$$\widetilde{u}^G = (1 - \alpha_G)\,\mathrm{Norm}(g^G) + \alpha_G u^G, \widetilde{u}^C = (1 - \alpha_C)\,\mathrm{Norm}(g^C) + \alpha_C u^C. \tag{6}$$

Channel-Level Policy. The channel-level policy score is obtained by aggregating incident weight-entry scores to each hidden channel. If δ^C_j is the channel correction for hidden channel j,

it is expanded to all incident weight entries before the parameter score update: $\widehat{\ell}^P = \ell^P + \gamma_P \operatorname{Expand}_C(\delta^C)$. This keeps the differentiable score path entrywise while the channel memory provides cross-context policy corrections.

4.2.2 Unified Graph-Channel Optimization

Coupled Score Generation. OUGP is not a combination of independent graph pruning and channel pruning. Instead, it jointly optimizes the two coupled sparsification dimensions through hidden-state utility while maintaining separate graph and channel memories. Each side receives the current keep-rate context of the other side. For graph unit $e \in \mathcal{E}$ and shared channel j, context encoders produce base scores $b_e^G = \phi_G(c_e^G), b_j^C = \phi_C(c_j^C)$, where c_e^G contains endpoint features, endpoint degrees, and a channel keep-rate summary, while c_j^C contains channel statistics, the channel identifier, the current channel score, and a graph keep-rate summary. In the primary OUGP setting, this cross-context pathway is enabled; the corresponding no-cross variant disables it as an ablation.

Cross-Context Memory Read. The graph memory S^G and the shared channel memory S^C use learned query/key/value projections. Their residual reads are $r_e^G = S^G q_e^G$ and $r_j^C = S^C q_j^C$, and the corrected scores are $\mathbb{G}_e = b_e^G + \gamma_G w_G^\top r_e^G$ and $\mathbb{C}_j = b_j^C + \gamma_C w_C^\top r_j^C$. The graph and channel contexts include the current channel and graph keep-rate summaries, respectively. Thus, the two ranking problems exchange capacity information through their context encoders before memory correction.

Residual Memory Update. For a memory batch with projected keys k_i and values v_i, the existing residual Q/K/V write uses the normalized utility $\widehat{u}_i$ to update the state according to

$$S \leftarrow \lambda S + \frac{\beta}{B} \sum_{i=1}^{B} \left[v_i \tanh\big(\widehat{u}_i - h_U(k_i S^\top)\big) - k_i S^\top \right] k_i^\top, \tag{7}$$

with the channel memory updated analogously for each channel. The implementation also retains the existing event and recall corrections. During source training, soft budgeted masks preserve gradient flow, and the source objective is

$$\mathcal{L}_{\text{src}} = \mathcal{L}_{\text{task}} + \lambda_{\text{sp}} \mathcal{R}_{\text{mask}}(\widetilde{\boldsymbol{G}}, \widetilde{\boldsymbol{C}}) + \lambda_{\text{bud}} \mathcal{R}_{\text{resource}}(\widetilde{\boldsymbol{G}}, \widetilde{\boldsymbol{C}}), \tag{8}$$

where $\widetilde{\boldsymbol{G}}$ and $\widetilde{\boldsymbol{C}}$ denote the soft budgeted masks used during source training, $\mathcal{R}_{\text{mask}}$ penalizes deviations of their mean keep rates from the graph and channel targets, and $\mathcal{R}_{\text{resource}}$ penalizes deviations of the normalized message and parameter costs from the configured resource budget. The resource term is omitted when its coefficient is set to zero.

Hardening and Materialization. After source training, $(\boldsymbol{G}, \boldsymbol{P}) = \operatorname{Harden}(\widehat{\ell}^G, \widehat{\ell}^P; \rho_G, \rho_P)$. Discarded edges and parameter entries are removed, and the compact GCN is frozen.

4.3 Energy-Shifted Parameter-Mask Test-Time Adaptation

Deployment Robustness. Pruning can increase sensitivity to feature and domain shifts. We therefore use unlabeled input-energy statistics to adapt parameter support while preserving source OUGP training. Each target view undergoes energy extraction, shift estimation, score reweighting, and fixed-budget mask materialization.

4.3.1 Channel-wise Input-Energy Shift

Input Energy Matrix. For a node-feature matrix $X \in \mathbb{R}^{N \times C}$, we compute the channel-wise input energy before graph propagation, We then measure the standardized channel-wise shift and its global adaptation strength:

$$\mathcal{E}_{ic}(X) = x_{ic}^2, \quad \mu_c^D = \frac{1}{N_D} \sum_{i=1}^{N_D} \mathcal{E}_{ic}(X_D), \quad \sigma_c^S = \sqrt{\frac{1}{N_S} \sum_{i=1}^{N_S} \big(\mathcal{E}_{ic}(X_S) - \mu_c^S\big)^2}, \quad D \in \{S, T\}. \tag{9}$$

$$z_c = \frac{\mu_c^T - \mu_c^S}{\sigma_c^S + \epsilon}, \qquad g = \tanh\left(\frac{\|z\|_2}{\sqrt{C}}\right). \tag{10}$$

Here, z captures the direction and relative magnitude of the channel-wise shift, while $g \in [0, 1)$ converts its RMS magnitude into a bounded adaptation strength. Target statistics are computed directly from deployment-time features and require no label information.

4.3.2 Energy-Guided Parameter-Mask Reweighting

Channel-to-Parameter Signal. The input shift is routed to hidden channels using the absolute first-layer weights. For hidden channel j, the normalized routing coefficient from input channel c is proportional to $|W_{jc}^{(1)}|$, giving

$$z_j^H = \sum_c \frac{|W_{jc}^{(1)}|}{\sum_{c'} |W_{jc'}^{(1)}| + \epsilon} z_c. \tag{11}$$

The resulting z_j^H transfers the input-domain shift into the hidden representation space. For a parameter entry i, its shift signal r_i is assigned according to the two channels connected by that entry: $r_i = z_c$ for an input-to-hidden entry, $r_i = (z_j^H + z_k^H)/2$ for a hidden-to-hidden entry, and $r_i = z_j^H$ for a hidden-to-output entry. We normalize the complete parameter signal by its RMS magnitude:

$$\widetilde{z}_i = \frac{r_i}{\sqrt{M^{-1} \sum_{j=1}^M r_j^2 + \epsilon}}. \tag{12}$$

Parameter-Score Update. Let s_i^S be the source OUGP parameter score and let λ_{pm} control the mask adaptation strength. The source score supplies the baseline ranking, while the gated target shift supplies an input-conditioned correction:

$$s_i^T = s_i^S + \lambda_{\text{pm}}\, g\, \text{Std}(s^S)\, \widetilde{z}_i. \tag{13}$$

Here, g controls the correction magnitude and the sign of $\widetilde{z}_i$ determines whether a parameter moves upward or downward in the ranking. The target parameter mask inherits the source keep budget and selects the highest-scoring entries:

$$m_i^T = \mathbf{1}\left[s_i^T \in \text{TopK}(s^T, K)\right], \qquad K = \sum_i m_i^S. \tag{14}$$

Thus, the parameter keep count and pruning ratio are preserved, while the retained support is reselected according to the unlabeled target shift.

4.3.3 Forward-Only Target Deployment

Target Forward. For the current target feature matrix X_T, the energy statistic, channel shift, and parameter scores are evaluated before message passing. The resulting mask is materialized once for the target view and applied to the source-trained weight tensors throughout the target forward:

$$\widehat{Y}_T = f\left(X_T, A_T; \{W_l \odot m_l^T\}_{l=1}^L\right). \tag{15}$$

Inference Path. The deployment state consists of the frozen compact GCN, the cached source energy statistics, the source parameter scores, and the target-view mask. The prediction path applies the reselected parameter support, performs the standard graph propagation, and produces logits through the original classifier. Consequently, each target view is handled by a label-free, forward-only evaluation whose adaptation signal is supplied by the current feature-energy profile.

5 Experiments

5.1 Experimental Setup

We evaluate OUGP on node classification benchmarks (Sen et al., 2008; Fey & Lenssen, 2019; Hu et al., 2020). We compare with representative graph pruning, model/channel pruning, and unified

pruning baselines, including NeuralSparse (Zheng et al., 2020), SGCN (Li et al., 2020), AdaptiveGCN (Li et al., 2021), LSP-P (Kosman et al., 2022), DSpar (Liu et al., 2023), GraSP (Wang et al., 2020), UGS (Chen et al., 2021), CGP (Liu et al., 2024), ACE-GLT (Wang et al., 2023), and UniFews (Liao et al., 2025).

All experiments use the same 2-layer GCN backbone with a hidden dimension of 32, identical data splits, pruning budgets, and random seeds unless otherwise specified. We adopt a label-stratified 2:1:1 train/validation/test split and evaluate performance using test accuracy and relative FLOPs reduction. For pruning experiments, both graph-edge and model-side sparsity are controlled under the same budget to ensure fair comparison across different pruning paradigms.

5.2 Experimental Results

Accuracy and Computational Efficiency. Under a fixed pruning budget of 30% graph sparsity and 30% parameter sparsity, Table 1 shows that OUGP+ESPM consistently reduces training FLOPs while maintaining competitive accuracy with the dense GCN. It achieves 0.92 and 1.10 percentage-point improvements on Cora and PubMed, respectively, with 22–25% FLOPs reductions across datasets. These results demonstrate that jointly optimizing graph and parameter pruning can achieve an effective accuracy-efficiency trade-off.

Table 1: Test accuracy and training FLOPs reduction under the label-stratified 2:1:1 protocol. All rows are averaged over five runs and approximately 30% sparsity in each supported pruning dimension. GP, MP, and TTA denote graph pruning capability, model-side parameter pruning, and test-time adaptation. The best and second-best results are highlighted with bold and underline, respectively. More details are provided in AppendixB.

Methods	GP	MP	TTA	Cora		PubMed		Cora-full		DBLP		ACM	
				$\mathcal{A}$	$\mathcal{F}_{\text{train}}$	$\mathcal{A}$	$\mathcal{F}_{\text{train}}$	$\mathcal{A}$	$\mathcal{F}_{\text{train}}$	$\mathcal{A}$	$\mathcal{F}_{\text{train}}$	$\mathcal{A}$	$\mathcal{F}_{\text{train}}$
Dense	✗	✗	✗	87.04	1.00×	86.36	1.00×	<u>70.35</u>	1.00×	**85.51**	1.00×	<u>92.58</u>	1.00×
UGS	✓	✓	✗	80.49 ↓6.55	1.71× ↑0.71	85.84 ↓0.52	1.71× ↑0.71	63.34 ↓7.01	1.71× ↑0.71	81.70 ↓3.81	1.71× ↑0.71	90.96 ↓1.62	1.71× ↑0.71
CGP	✓	✓	✗	85.79 ↓1.25	0.85× ↓0.15	87.37 ↑1.01	0.85× ↓0.15	68.37 ↓1.98	0.85× ↓0.15	83.60 ↓1.91	0.85× ↓0.15	90.31 ↓2.27	0.85× ↓0.15
UniFews	✓	✓	✗	82.16 ↓4.88	0.86× ↓0.14	84.02 ↓2.34	0.86× ↓0.14	58.94 ↓11.41	0.85× ↓0.15	84.00 ↓1.51	0.85× ↓0.15	79.82 ↓12.76	0.85× ↓0.15
ACE-GLT	✓	✓	✗	76.77 ↓10.27	2.56× ↑1.56	85.24 ↓1.12	2.56× ↑1.56	66.97 ↓3.38	2.56× ↑1.56	84.28 ↓1.24	2.56× ↑1.56	91.82 ↓0.76	2.56× ↑1.56
DSpar	✓	✗	✗	86.16 ↓0.88	0.97× ↓0.03	85.85 ↓0.51	0.96× ↓0.04	68.24 ↓2.10	0.95× ↓0.05	84.80 ↓0.72	0.95× ↓0.05	92.55 ↓0.03	0.97× ↓0.03
SGCN	✓	✗	✗	<u>87.44</u> ↑0.41	1.15× ↑0.15	85.61 ↓0.75	1.15× ↑0.15	67.51 ↓2.83	1.15× ↑0.15	84.40 ↓1.12	1.15× ↑0.15	91.95 ↓0.63	1.15× ↑0.15
AdaptiveGCN	✓	✗	✗	85.90 ↓1.14	0.97× ↓0.03	85.91 ↓0.45	0.96× ↓0.04	68.72 ↓1.63	0.95× ↓0.05	84.63 ↓0.89	0.95× ↓0.05	92.28 ↓0.30	0.97× ↓0.03
LSP-P	✓	✗	✗	86.05 ↓0.99	0.97× ↓0.03	86.08 ↓0.28	0.96× ↓0.04	69.78 ↓0.57	0.95× ↓0.05	85.20 ↓0.31	0.95× ↓0.05	92.45 ↓0.13	0.97× ↓0.03
NeuralSparse	✓	✗	✗	86.41 ↓0.63	0.97× ↓0.03	85.84 ↓0.52	0.96× ↓0.04	69.04 ↓1.30	0.95× ↓0.05	84.72 ↓0.79	0.95× ↓0.05	91.42 ↓1.15	0.97× ↓0.03
GraSP	✗	✓	✗	85.38 ↓1.66	**0.73**× ↓0.27	85.48 ↓0.88	**0.73**× ↓0.27	52.55 ↓17.79	**0.75**× ↓0.25	85.19 ↓0.33	**0.74**× ↓0.26	91.69 ↓0.89	**0.73**× ↓0.27
GCNP	✗	✓	✗	83.80 ↓3.24	0.85× ↓0.15	**87.69** ↑1.33	0.83× ↓0.17	57.95 ↓12.40	0.82× ↓0.18	83.36 ↓2.16	0.87× ↓0.13	89.21 ↓3.37	0.87× ↓0.13
OUGP+ESPM (Ours)	✓	✓	✓	**87.96** ↑0.92	<u>0.75</u>× ↓0.25	<u>87.46</u> ↑1.10	<u>0.78</u>× ↓0.22	**68.02** ↓2.33	<u>0.76</u>× ↓0.24	<u>85.44</u> ↓0.07	<u>0.76</u>× ↓0.24	**93.07** ↑0.49	<u>0.75</u>× ↓0.25

Deployment Robustness. Beyond clean evaluation, we further investigate whether compact GNNs remain reliable under distribution shifts. Different from standard in-distribution testing, OOD evaluation measures the capability of a pruned model to maintain predictive performance when the deployment distribution differs from the source training distribution. Please refer to Table 4.

5.3 Diagnostic Analysis

Key Components. We perform component-wise ablation studies to analyze the contribution of individual modules in OUGP. Specifically, we evaluate the effects of the memory mechanism and Layer-wise Hidden Coupling Utility (LHCU) while keeping the remaining optimization pipeline unchanged throughout the ablation experiments. Table 2 shows that both the memory mechanism and LHCU contribute to the final performance. Removing either component leads to a performance drop, while using both components achieves the highest accuracy on both Cora and Photo. The improvement is particularly pronounced on Photo. Adding the memory module increases the accuracy from 77.66% to 81.87%, demonstrating its effectiveness in preserving useful pruning information. Starting from the memory-enabled model, further incorporating LHCU improves the accuracy to 83.07%, indicating that hidden-state coupling provides additional benefits by capturing interactions between graph and parameter pruning decisions.

Energy-guided Parameter Mask Adaptation. We further compare different adaptation strategies under the same materialized compact GNN. During adaptation, the backbone parameters and graph structure remain frozen, while ESPM adjusts the parameter mask according to unlabeled target-domain statistics. Table 3 compares ESPM with representative test-time adaptation strategies under the same compact backbone. ESPM achieves 84.90% and 84.55% accuracy on Cora and PubMed, respectively, substantially outperforming Tent, EATA, and SAR. These results demonstrate that ESPM provides a lightweight alternative to optimization-based TTA by adapting parameter supports through unlabeled target statistics while preserving the frozen compact backbone.

Table 2: Ablation study of key components in OUGP. We evaluate the effects of the memory module and Layer-wise Hidden Coupling Utility (LHCU).

Memory	LHCU	Cora $\mathcal{A}$	Photo $\mathcal{A}$
✗	✗	85.46 ± 0.78	77.66 ± 1.68
✗	✓	85.49 ± 0.82	77.59 ± 1.76
✓	✗	85.94 ± 1.21	81.87 ± 3.07
✓	✓	$\mathbf{86.16 \pm 1.21}$	$\mathbf{83.07 \pm 0.97}$

Table 3: Comparison of test-time adaptation strategies on compact GNNs. All methods adapt the same frozen compact backbone under deployment shifts.

Method	Cora $\mathcal{A}$	PubMed $\mathcal{A}$
Tent	34.65 ± 1.06	41.06 ± 1.05
EATA	35.09 ± 9.29	46.92 ± 11.71
SAR	48.34 ± 3.22	55.38 ± 4.52
ESPM	$\mathbf{84.90 \pm 0.64}$	$\mathbf{84.55 \pm 0.96}$

Hyper-Parameters. We further study the influence of pruning ratios and memory ranks to evaluate the stability of OUGP under different compression settings.

5.4 Performance Stability under Varying Weight and Graph Sparsity

To assess the sensitivity of different pruning strategies to compression intensity, we vary weight sparsity while fixing graph sparsity at 30%, and vary graph sparsity while fixing weight sparsity at 30%. We evaluate UGS, ACE-GLT, and OUGP on Cora and CiteSeer. As shown in Fig. 4, performance remains relatively stable as weight sparsity increases. Under higher graph sparsity, UGS and ACE-GLT show more pronounced accuracy degradation, whereas OUGP maintains comparatively stable performance and achieves the highest accuracy across the evaluated settings.

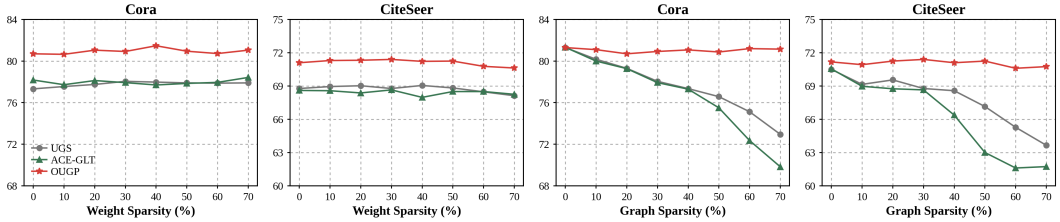


Figure 4: Test accuracy on Cora and CiteSeer under varying weight sparsity (left two panels; graph sparsity fixed at 30%) and graph sparsity (right two panels; weight sparsity fixed at 30%).

6 Conclusion

In this paper, we propose Online Unified Graph Pruning for compact graph neural networks. OUGP jointly optimizes graph-edge sparsity and parameter sparsity through hidden-channel-guided feedback, enabling unified pruning and producing a materialized compact GNN with fixed graph and parameter masks. To improve adaptability under deployment shifts, we further introduce an energy-shift-guided parameter-mask test-time adaptation module, which adjusts parameter selection according to unlabeled target-domain statistics while preserving the compact structure. Extensive experiments demonstrate that OUGP achieves a superior accuracy-efficiency trade-off and maintains robust performance under deployment shifts compared with existing pruning methods, highlighting the effectiveness of hidden-state-guided unified pruning for efficient and reliable compact GNN.

AI USE STATEMENT

AI tools were used to assist with manuscript writing and language polishing, as well as to support code review and implementation correctness checks. They were not used for data analysis or interpretation of experimental results. All experimental analyses and conclusions were conducted and verified by the authors, who take full responsibility for the content of this work.

REPRODUCIBILITY STATEMENT

We provide the code and experimental scripts in an anonymous repository to support reproducibility. The experimental setup, including dataset splits, model configurations, pruning settings, training settings, and evaluation protocols, is documented in the main paper and the appendix. These materials are intended to enable independent reproduction and verification of the reported results.

REFERENCES

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1695–1706. PMLR, 2021.

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 and Data Mining*, pp. 257–266, 2019. doi: 10.1145/3292500.3330925.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning*, pp. 3294–3304, 2021.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, 2020.

Di Jiang, Yuan Cao, and Qiang Yang. On the channel pruning using graph convolution network for convolutional neural network acceleration. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 3107–3113, 2022. doi: 10.24963/ijcai.2022/431.

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

Eitan Kosman, Joel Oren, and Dotan Di Castro. Lsp: Acceleration of graph neural networks via locality sensitive pruning of graphs. In *2022 IEEE International Conference on Data Mining Workshops*, pp. 690–697, 2022. doi: 10.1109/ICDMW58026.2022.00095.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

Dongyue Li, Tao Yang, Lun Du, Zhezhi He, and Li Jiang. Adaptivegcn: Efficient gcn through adaptively sparsifying graphs. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, pp. 3206–3210. ACM, 2021. doi: 10.1145/3459637.3482049.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017.

Jiayu Li, Tianyun Zhang, Hao Tian, Shengmin Jin, Makan Fardad, and Reza Zafarani. Sgcn: A graph sparsifier based on graph convolutional networks. In *Advances in Knowledge Discovery and Data Mining*, pp. 275–287. Springer, 2020. doi: 10.1007/978-3-030-47426-3_22.

Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning*, pp. 6028–6039, 2020.

Ningyi Liao, Zihao Yu, Ruixiao Zeng, and Siqiang Luo. Unifews: You need fewer operations for efficient graph neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 37587–37609. PMLR, 2025.

Chuang Liu, Xueqi Ma, Yibing Zhan, Liang Ding, Dapeng Tao, Bo Du, Wenbin Hu, and Danilo P. Mandic. Comprehensive graph gradual pruning for sparse training in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 35(10):14903–14917, 2024. doi: 10.1109/TNNLS.2023.3282049.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shougang Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. Dspar: An embarrassingly simple strategy for efficient gnn training and inference via degree-based sparsification. *Transactions on Machine Learning Research*, 2023.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16888–16905. PMLR, 2022.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Zhiquan Wen, Yaofo Chen, Peilin Zhao, and Mingkui Tan. Towards stable test-time adaptation in dynamic wild world. In *International Conference on Learning Representations*, 2023.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, 2020.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 9229–9248, 2020.

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

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021.

Yuwen Wang, Shunyu Liu, Kaixuan Chen, Tongtian Zhu, Ji Qiao, Mengjie Shi, Yuanyu Wan, and Mingli Song. Adversarial erasing with pruned elements: Towards better graph lottery tickets. In *European Conference on Artificial Intelligence*, pp. 2583–2590, 2023. doi: 10.3233/FAIA230564.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pp. 6861–6871, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

Marvin Zhang, Sergey Levine, and Chelsea Finn. Memo: Test time robustness via adaptation and augmentation. In *Advances in Neural Information Processing Systems*, 2022.

Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust graph representation learning via neural sparsification. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11458–11468. PMLR, 2020.

A Details of the Comparison Methods

The comparison methods in the main experiments are grouped by the part of the GNN that they sparsify. All matched rows use the same backbone, data split, training budget, and evaluation mask unless stated otherwise. Methods in the matched tables share the protocol summarized in Table 6. Repository-local proxies and official-code screens are identified explicitly because they do not necessarily have the same mask semantics as the materialized structured controls.

A.1 Pruning methods and component variants

- **Dense** (Kipf & Welling, 2017): The unpruned 2-layer GCN. It is the reference for accuracy and relative computational cost.
- **DropEdge** (Rong et al., 2020): DropEdge randomly removes a subset of graph edges during training to reduce graph density and mitigate over-smoothing in GNNs. By perturbing the graph structure across training iterations, it regularizes message passing while reducing the computational cost of graph propagation.
- **UniFews** (Liao et al., 2025): UniFews jointly sparsifies graph propagation and weight transformation through entry-wise pruning. It progressively increases sparsity across GNN layers and enables on-the-fly simplification of graph and weight operations during training and inference.
- **ACE-GLT** (Wang et al., 2023): Adversarial Complementary Erasing (ACE) explores useful information from pruned graph edges and model weights to construct improved Graph Lottery Tickets (GLTs). It complements iterative magnitude-based pruning by reconsidering previously pruned components during the ticket-search process, thereby recovering potentially useful structural and parameter information that may be discarded.
- **DSpar** (Liu et al., 2023): DSpar sparsifies graphs before training using node degree information as a lightweight proxy for edge importance. It removes redundant edges with negligible preprocessing overhead and applies the resulting sparsified graph throughout training and inference.
- **SGCN** (Li et al., 2020): Sparsified Graph Convolutional Network (SGCN) formulates graph sparsification as an optimization problem and solves it using the Alternating Direction Method of Multipliers (ADMM). It learns a sparsified graph by pruning edges while preserving the performance of downstream GNN models.
- **AdaptiveGCN** (Li et al., 2021): AdaptiveGCN uses a supervised edge predictor to learn layer-specific edge selection strategies from downstream task feedback. It adaptively removes task-irrelevant edges during training and performs inference using the selected sparse graphs.
- **LSP-P** (Kosman et al., 2022): Locality-Sensitive Pruning (LSP) uses locality-sensitive hashing and random projections to prune graph edges while preserving similarities between local graph environments. LSP-P applies random-projection-based hashing to construct a consistently sparsified graph as a preprocessing step.
- **NeuralSparse** (Zheng et al., 2020): NeuralSparse learns to remove potentially task-irrelevant graph edges using a supervised neural sparsification model. The sparsification process incorporates both structural and node-feature information and is optimized through downstream task feedback.
- **GraSP** (Wang et al., 2020): Gradient Signal Preservation (GraSP) identifies sparse subnetworks before training by preserving gradient flow through the network. It prunes connections at initialization according to their contribution to gradient preservation, enabling sparse training without first training a dense model.
- **GCNP** (Jiang et al., 2022): GCNP accelerates GNN inference through structured channel pruning. It uses a LASSO-based formulation to identify important hidden feature dimensions and removes redundant channels to reduce computation and memory requirements for large-scale real-time inference.
- **SNIP** (Lee et al., 2019): Single-shot Network Pruning (SNIP) identifies important connections at initialization based on their sensitivity to removal. By pruning less sensitive connections before training, SNIP obtains a sparse subnetwork in a single step without requiring iterative pruning and retraining.

- **UGS** (Chen et al., 2021): Unified GNN Sparsification (UGS) jointly sparsifies the graph structure and GNN parameters within a unified end-to-end optimization framework. Specifically, it introduces separate learnable masks for the graph adjacency matrix and model weights, and jointly optimizes them to identify a compact graph-model pair while maintaining task performance under sparsity constraints.
- **CGP** (Liu et al., 2024): Comprehensive Graph Gradual Pruning (CGP) proposes a dynamic during-training pruning framework for GNNs. It comprehensively co-sparsifies graph structures, node feature dimensions, and model parameters with a single training process, while incorporating a regrowth mechanism to restore pruned connections. This design reduces training and inference costs without requiring iterative retraining.

A.2 Test-time adaptation variants

- **Frozen compact model**: The materialized compact GCN is evaluated with its source-hardened graph and parameter supports. This provides the frozen reference for measuring ESPM gains.
- **Energy-shifted parameter-mask adaptation**: The target input-energy profile is compared with cached source statistics to obtain a channel-wise shift, which is routed to parameter entries through the source-trained weight structure.
- **Parameter-score re-ranking**: The source parameter scores provide the baseline ranking, while the energy-conditioned correction re-ranks parameter entries under the original keep budget.
- **Target-specific mask**: The re-ranked scores materialize a target-specific parameter mask before message passing, preserving the source parameter keep rate.
- **Forward-only deployment**: The target feature-energy statistics determine the parameter support before message passing, after which the compact GCN produces predictions through its original classifier.
- **Budget-preserving adaptation**: The graph structure and parameter values remain fixed, while the distribution of retained parameter entries follows the current deployment shift.

A.3 OOD Evaluation Protocol.

In contrast to the IID setting, OOD evaluation uses the full target graph and does not perform target-side graph pruning. Therefore, graph-only pruning methods, including DSpar, SGCN, AdaptiveGCN, LSP-P, and NeuralSparse, cannot activate their graph pruning mechanisms on the target domain, making their results not directly comparable under this protocol. We consequently exclude these five methods from the OOD comparison and retain methods with applicable model-side pruning and/or test-time adaptation capabilities. The ood results are in Table 4.

Table 4: Directional OOD transfer accuracy on DBLPv8 → ACMv9 and ACMv9 → DBLPv8. Each direction reports source-domain accuracy, target-domain accuracy, and Avg Rank. All rows are averaged over at least five runs. GP, MP, and TTA denote graph pruning capability, model-side pruning, and test-time adaptation. Avg Rank is computed as $\mathrm{AvgRank}_i^d = \frac{R_i^{d,\mathrm{src}} + R_i^{d,\mathrm{tgt}}}{2}$, where $R_i^{d,\mathrm{src}}$ and $R_i^{d,\mathrm{tgt}}$ are the ranks of method i on source and target accuracy, respectively (1 is best). For each metric, the best and second-best results are highlighted with bold and underline, respectively. Please see Sec D.1

Methods	GP	MP	TTA	DBLPv8 → ACMv9			ACMv9 → DBLPv8		
				Source acc	Target acc	Avg Rank	Source acc	Target acc	Avg Rank
UGS	✓	✓	✗	98.03±0.30	69.53±2.70	3.0	80.63±0.86	54.43±4.34	6.0
CGP	✓	✓	✗	90.80±0.70	<u>70.44±1.00</u>	4.5	81.06±0.72	62.54±1.90	4.5
UniFews	✓	✓	✗	96.92±0.41	64.11±0.73	5.5	78.81±1.20	**70.92±2.36**	4.0
ACE-GLT	✓	✓	✗	97.61±0.29	65.70±4.34	4.5	83.02±0.35	57.62±5.33	4.5
GraSP	✗	✓	✗	**98.42±0.15**	69.01±1.94	<u>2.5</u>	**83.84±0.67**	64.47±8.65	<u>2.5</u>
GCNP	✗	✓	✗	96.70±1.17	56.80±3.83	6.5	80.39±1.07	66.41±4.72	4.5
OUGP+ESPM (Ours)	✓	✓	✓	<u>98.06±0.58</u>	**71.46±1.84**	**1.5**	<u>83.81±0.76</u>	<u>67.69±3.41</u>	**2.0**

B Evaluation Metrics

Training FLOPs Ratio. The reported training FLOPs ratio only accounts for the source training cost and excludes the deployment-time ESPM overhead, as existing baselines do not involve test-time adaptation.

C Dataset and Experimental Protocol

The main comparison covers five dataset groups: Cora, PubMed, Cora-full, DBLP, and ACM. Cora-full denotes the CitationFull-Cora graph and DBLP denotes the CitationFull-DBLP graph. All datasets are evaluated as node classification problems under the label-stratified 2:1:1 protocol. The same test mask is used for all methods within a dataset. Please refer to Table 6.

Table 5: Dataset coverage and representative clean results. Accuracy values are reproduced from the main comparison to make the five dataset groups explicit in the appendix. The second quantity is the relative reported training-cost factor used in the main table.

Method	Cora		PubMed		Cora-full		DBLP		ACM	
	Acc. (%)	F_{train}	Acc. (%)	F_{train}	Acc. (%)	F_{train}	Acc. (%)	F_{train}	Acc. (%)	F_{train}
Dense	87.04	$1.00\times$	86.36	$1.00\times$	70.35	$1.00\times$	85.51	$1.00\times$	92.58	$1.00\times$
OUGP+ESPM (Ours)	**87.96**	$0.75\times$	**87.46**	$0.78\times$	**68.02**	$0.76\times$	**85.44**	$0.76\times$	**93.07**	$0.75\times$

Table 6: Common experimental settings for the matched comparison.

Item	Setting
Backbone	2-layer GCN with a shared hidden representation
Task	Transductive node classification
Dataset split	Label-stratified 2:1:1 train/validation/test split
Pruning budget	Approximately 30% graph and 30% model-side sparsity
Evaluation	Accuracy on the predefined test mask
Adaptation	Forward-only energy-shifted parameter-mask re-ranking at deployment

D Additional Experimental Results

D.1 Results on OOD Transfer for Citation Networks

Table 4 reports directional OOD transfer results between DBLPv8 and ACMv9. For DBLPv8 $\rightarrow$ ACMv9, OUGP+ESPM achieves the highest target accuracy of 71.46%, with the best Avg Rank of 1.5. For ACMv9 $\rightarrow$ DBLPv8, UniFews achieves the highest target accuracy of 70.92%, consistent with its message-norm-aware entry-wise sparsification and progressive layer-wise sparsity. OUGP+ESPM ranks second with 67.69% and an Avg Rank of 2.0. Notably, it achieves 3.22 percentage points higher target accuracy than GraSP despite nearly identical source accuracy (83.81% vs. 83.84%). Overall, OUGP+ESPM ranks first and second in target accuracy across the two directions, demonstrating robust OOD transfer performance.

D.2 Hidden-state Coupling across Layers.

To understand the coupling between graph and channel pruning, we analyze the layer-wise hidden-state changes. Figure 6 shows that the channel-induced perturbation $\|\Delta H_C^{(l)}\|_F$ grows rapidly as the network becomes deeper. This phenomenon suggests that early-layer channel removal can propagate and amplify through subsequent message-passing layers. Moreover, the non-additive interaction ratio between graph and channel pruning becomes more apparent in deeper layers, revealing that the two compression dimensions affect shared hidden representations in a coupled manner. Therefore, independently optimizing graph sparsity and channel sparsity may overlook such layer-wise interactions and result in suboptimal compression decisions.

D.3 Component ablation

The component ablation evaluates the contribution of the memory mechanism and LHCU on Cora and Photo. The table is included directly from the project source so that the appendix remains self-contained.

D.4 Test-time adaptation comparison

The adaptation comparison reports representative ESPM baselines on Cora and PubMed. These values are interpreted as a robustness comparison under test-time shift, not as a claim that one adaptation rule dominates on all datasets.

D.5 Visualization results

To provide a clearer view of OUGP's cross-domain behavior and the hidden-state effects of joint pruning, we visualize bidirectional OOD transfer and the interaction between graph and channel pruning. For the hidden-state analysis, we examine the non-additive interaction ratio during training, the final hidden-state magnitude and effective rank, and the layer-wise channel-induced perturbation and graph–channel coupling across Cora, CiteSeer, Photo, PubMed, DBLP, and ACM. For cross-domain transfer, we compare DBLPv8 $\rightarrow$ ACMv9 and ACMv9 $\rightarrow$ DBLPv8 under frozen OUGP and energy-shifted parameter-mask deployment.

D.6 Dataset-dependent Hidden-state Coupling.

We further analyze the evolution and dataset dependency of hidden-state coupling during pruning. As shown in Figure 5, the interaction ratio consistently increases throughout training across different datasets, indicating that the observed coupling between graph and channel pruning emerges and becomes more prominent during optimization rather than being a static property of the initialized model. This observation suggests that the two pruning dimensions jointly influence the learned hidden representations.

Meanwhile, the final interaction strength varies substantially across datasets. Different graph structures and feature distributions exhibit different levels of non-additive interactions, suggesting that the effect of jointly considering graph and channel pruning depends on dataset characteristics. Therefore, treating graph sparsification and channel pruning as independent objectives may overlook dataset-specific coupling patterns.

We further examine the hidden representation characteristics after pruning. The hidden-state magnitude and effective rank show different behaviors among datasets and pruning strategies, revealing that pruning affects not only the scale but also the intrinsic structure of learned representations. These variations demonstrate that compact GNNs exhibit diverse representation responses under different graph distributions, motivating pruning strategies that explicitly capture graph-channel interactions.

E Complexity Analysis

Let $N = |\mathcal{V}|$, $M = |\mathcal{E}|$, d_0 be the input dimension, d the hidden dimension, and C the number of classes. After hardening, OUGP retains M_{k} graph edges and a subset of parameter entries under the prescribed graph and parameter sparsity budgets. With self-loops, we denote $\bar{M} = M + N$ and $\bar{M}' = M_{\mathrm{k}} + N$.

Materialized GCN computation. For the evaluated 2-layer GCN, the dense model requires

$$\mathcal{C}_{\mathrm{dense}} = \bar{M}(d_0 + d) + N(d_0 d + dC). \tag{16}$$

After hardening, graph-edge and parameter masks are materialized into the compact model, yielding

$$\mathcal{C}_{\mathrm{mat}} = \bar{M}'(d_0 + d) + N(\mathrm{nnz}(W_1) + \mathrm{nnz}(W_2)), \tag{17}$$

where $\mathrm{nnz}(W_l)$ denotes the number of retained parameter entries in layer l. Thus, graph pruning reduces message-passing cost through $\bar{M}'$, while parameter pruning directly reduces the cost of the corresponding linear transformations.

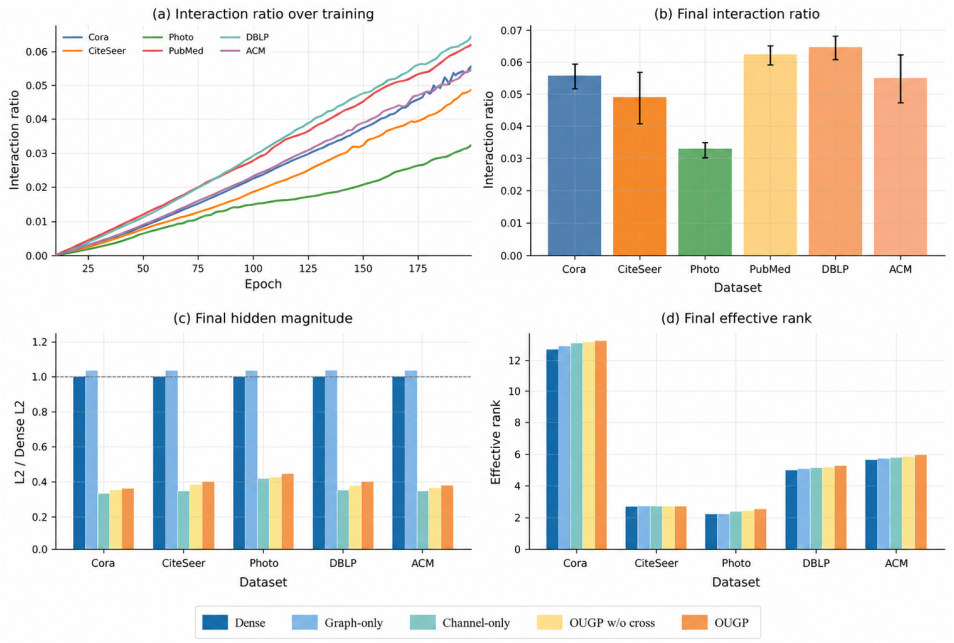


Figure 5: Hidden-state diagnostics of graph and channel pruning. The upper-left panel shows the non-additive interaction ratio during training. The upper-right panel compares the final interaction strength across datasets. The lower-left panel reports the final hidden-state magnitude relative to the dense model, while the lower-right panel shows the effective rank of the hidden representations.

OUGP training and memory overhead. During source training, OUGP jointly learns graph-edge and parameter-entry masks, with LHCU providing channel-level utility feedback that is expanded to incident parameter entries. The graph and parameter score variables require $O(M + P)$ storage, where P is the total number of prunable parameter entries. The graph and channel memory states introduce fixed-rank storage of

$$O(B_G r^2 + dr^2), \tag{18}$$

where r is the memory rank and B_G denotes the number of graph-memory branches. The final configuration uses $r = 8$, while optional event and recall buffers are disabled.

ESPM overhead. After source training and hardening, the compact GNN and its graph structure are frozen. ESPM computes channel-wise input-energy statistics from unlabeled target features and compares them with cached source statistics to obtain the channel-wise shift z. The shift is then routed to parameter entries and used to update their scores, followed by fixed-budget top-K re-ranking to obtain a target-specific parameter mask. For N target nodes, C input channels, and P parameter entries, energy-statistic computation requires $O(NC)$ operations, channel-to-parameter routing requires $O(P)$ operations, and top-K selection requires $O(P \log P)$ operations with a sorting-based implementation. No parameter values are updated during this process; the resulting parameter mask is directly applied to the frozen compact GNN for target inference.

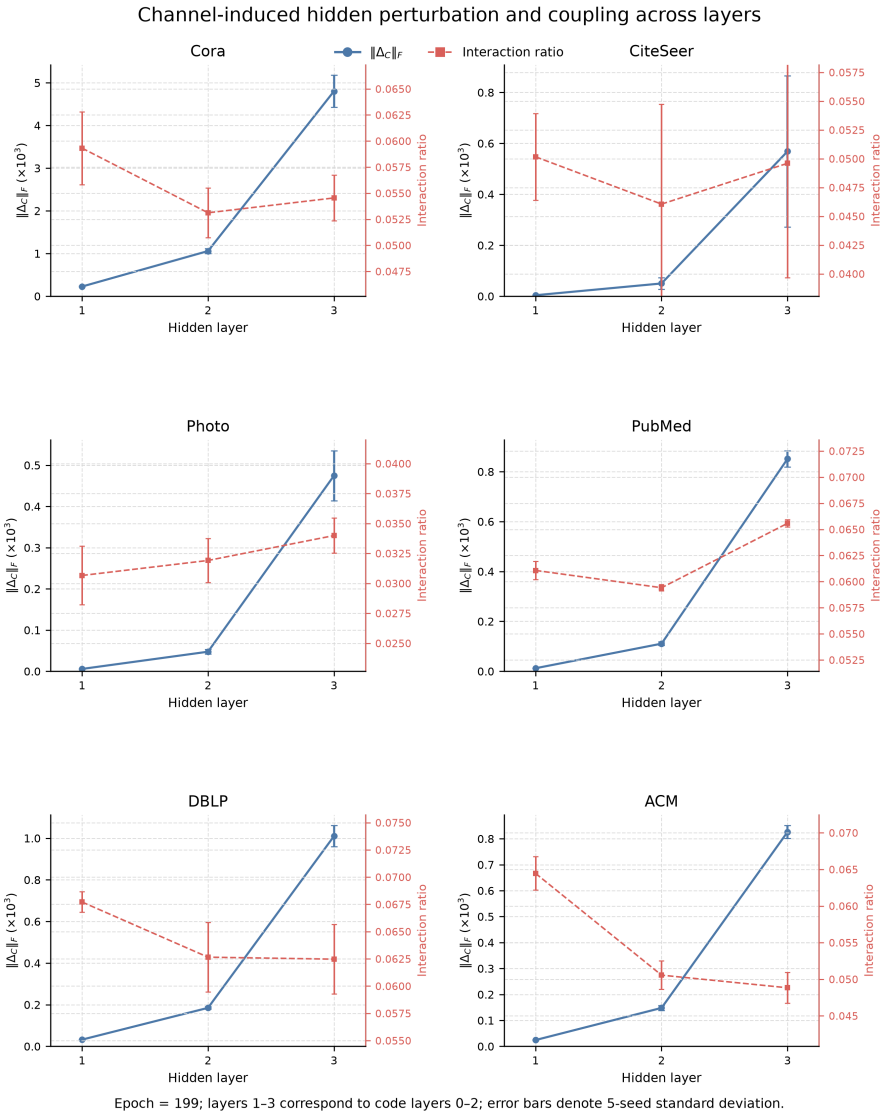


Figure 6: Layer-wise hidden-state coupling analysis. The blue curve shows the channel-induced hidden perturbation $\|\Delta_C^{(l)}\|_F$, while the red dashed curve reports the non-additive interaction ratio between graph pruning and channel pruning. The error bars denote the standard deviation over five seeds. Please see Sec D.2.