SEEING MORE, DISTINGUISHING LESS? LEARNING FROM GRAPH PROPAGATION STATES AND CHANGES

Anonymous authors
Paper under double-blind review

ABSTRACT

Decoupled graph neural networks precompute multi-hop features through graph diffusion, enabling information from different propagation scales to be used without increasing neural network depth. However, existing cross-hop fusion methods often assign a single weight to each propagation step and share it across feature dimensions, ignoring that different features could benefit from different propagation scales. Moreover, using more propagation scales does not necessarily improve node distinguishability, as combining multi-hop states can weaken useful differences across propagation steps. To this end, we propose **F**eature-wise Fusi**o**n a**c**ross Hops for **A**daptive Graph **L**earning (FOCAL), a graph learning framework that models Propagation States and Changes through feature-wise cross-hop fusion. In the method, Propagation States describe feature values after diffusion, while Propagation Changes capture their differences between consecutive diffusion steps. FOCAL learns independent weights for each feature in the State and Change branches, allowing different features to use different combinations of propagation operators and diffusion steps. Node-level gates further adjust the contributions of the encoded State and Change representations for each node. Experiments on eight node classification datasets show that FOCAL achieves the highest accuracy on seven datasets. Ablation studies and weight analysis further demonstrate the contributions of Propagation Changes, feature-wise fusion, separate State and Change weights, and Node gates.

1 INTRODUCTION

Graph neural networks (GNNs) have been widely used for learning from graph data such as social networks, citation networks, and molecular structures. Their applications include node classification (Kipf & Welling, 2017; Veličković et al., 2018; Hamilton et al., 2017), link prediction (Zhang & Chen, 2018; Zhang et al., 2021), and graph-level tasks (Xu et al., 2019; Ying et al., 2018; Dwivedi et al., 2023). Many GNN architectures follow the message-passing framework, which combines neighborhood aggregation with learnable feature transformations to update node representations (Gilmer et al., 2017; Scarselli et al., 2009; Wu et al., 2021). Repeating these operations allows nodes to use information from larger neighborhoods and produces node representations at different propagation scales.

In conventional message-passing architectures, increasing the propagation range generally requires stacking additional neural layers. However, repeated propagation can lead to over-smoothing, weakening distinctions between node representations and degrading classification performance (Li et al., 2018; Oono & Suzuki, 2020). Decoupled architectures separate graph propagation from feature transformation, allowing propagation depth to be chosen independently of neural network depth (Gasteiger et al., 2019; Wu et al., 2019). Some decoupled methods, such as SIGN and FSGNN, use fixed graph diffusion operators to precompute propagation states ($\{H^{(0)}, \ldots, H^{(K)}\}$) (Rossi et al., 2020; Maurya et al., 2021). These precomputed states provide information from different hops without requiring more neural network layers. However, decoupling propagation from feature transformation does not remove the challenge of multi-scale fusion. Given these multi-scale representations, the key question is how to fuse them without losing useful information for node classification.

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.

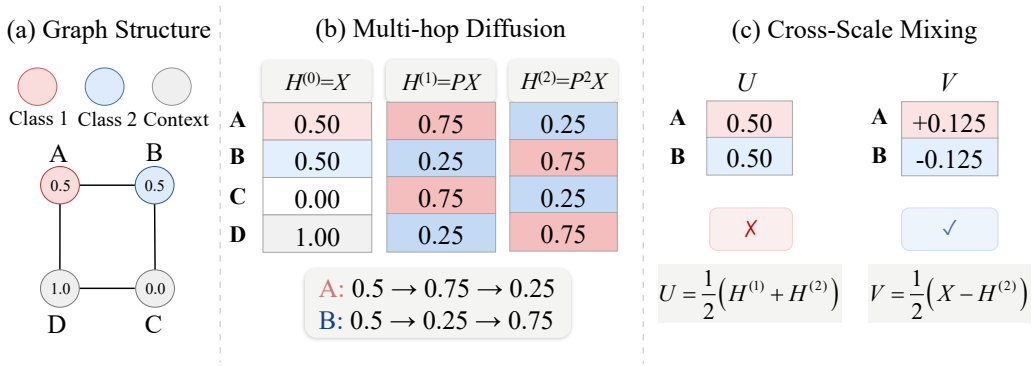


Figure 1: Change fusion keeps node differences lost in equal-weight State fusion. (a) Target nodes A and B have different labels but the same input value. (b) One-step and two-step graph diffusion assign different values to the two targets. (c) Equal-weight State fusion maps both targets to 0.50, while equal-weight Change fusion gives 0.125 and -0.125.

Existing methods address the above question in different ways. Some methods use fixed fusion weights, while others learn one weight for each propagation step (Chien et al., 2021). Another choice is to keep each hop representation independent, either by giving each one its own transformation or by concatenating them together (Xu et al., 2018; Abu-El-Haija et al., 2019). Fusing the states with one weight per hop keeps the representation size fixed, but applies the same scale mixture to every feature dimension. Different features could respond differently to graph diffusion. For some features, information from larger neighborhoods can help identify a node's class. For others, useful differences are already present at shallow hops and could become weaker after further propagation. A shared hop weight increases or decreases the contribution of that hop for all features together. It cannot give one feature more weight at shallow hops and another more weight at deeper hops. Cross-hop fusion should learn scale weights for each feature. Concatenation keeps the hop representations separate, but its feature dimension grows with the number of hops.

This raises a question as the number of propagation steps K increases: *do more propagation scales necessarily improve node distinguishability?* The answer depends on how these states are fused. Although each Propagation State may retain useful differences, combining them can weaken these differences, such that seeing more can lead to distinguishing less. The challenge is therefore not only to determine which propagation scales are useful for each feature, but also to capture how each feature varies across consecutive Propagation steps. This motivates us to consider Propagation Changes, defined as the previous state minus the current state at each diffusion step. States describe feature values after diffusion, while changes describe how these values change between consecutive steps. For the same feature, useful state values and useful changes may occur at different hops. State and Change fusion should learn their own scale weights.

Figure 1 illustrates a case in which propagation changes keep node differences that State fusion fails to preserve. Nodes A and B have different labels but the same input value. Their states differ at both propagation steps. Equal-weight State fusion gives the two nodes the same value, while equal-weight Change fusion gives different values. In this example, Change fusion keeps a distinction that is lost in the fused State representation. This motivates learning from both states and changes.

Motivated by the above observations, this paper proposes **F**eature-wise Fusi**o**n a**c**ross Hops for **A**daptive Graph **L**earning (FOCAL). FOCAL learns independent cross-hop weights for each feature in State and Change fusion. At the feature level, it learns independent cross-hop weights for each feature in State and Change fusion, allowing the same feature to use different propagation scales in the two branches. At the node level, gates adjust the contributions of the encoded State and Change representations, while the encoded Input representation is directly included in prediction. In this way, feature-wise weights determine which propagation scales contribute to each feature, while node-level gates determine how strongly each node uses the fused State and Change representations.

- We illustrate how cross-hop fusion can affect node distinctions. A simple graph example shows that Change fusion can keep useful differences lost in State fusion, motivating the joint use of states and changes.
- We propose FOCAL, which learns independent feature-wise scale weights for State and Change fusion. This design allows each feature to use different propagation steps for the two representations. Node-level gates further adjust their contributions to each node's prediction.
- We evaluate FOCAL on diverse node classification benchmarks covering both homophilic and heterophilic graphs under a unified experimental setting. The results demonstrate that FOCAL achieves strong and stable performance compared with representative graph learning methods.

2 Related Work

Graph diffusion and cross-hop fusion. Decoupled GNNs separate graph propagation from feature transformation. SGC removes intermediate nonlinear transformations and uses fixed propagation followed by a linear classifier (Wu et al., 2019). APPNP uses fixed propagation weights based on personalized PageRank (Gasteiger et al., 2019), while GPR-GNN learns the weights of different propagation steps (Chien et al., 2021). Other methods combine information from multiple layers or hops. Jumping Knowledge Networks fuse representations from different network layers (Xu et al., 2018), and MixHop combines features computed from different propagation steps within each layer (Abu-El-Haija et al., 2019). Among precomputation-based methods, SIGN transforms precomputed graph features through separate branches and concatenates their outputs (Rossi et al., 2020). FSGNN uses separate feature transformations and softmax-normalized selection weights for propagated feature matrices (Maurya et al., 2021). Each selection weight is a scalar shared by all feature dimensions in its corresponding matrix.

Neighborhood information. Previous studies combine different graph signals to model neighborhood information. FAGCN adaptively combines low- and high-frequency signals during message passing (Bo et al., 2021). ACM uses aggregation, diversification, and identity channels, and learns their mixing weights for each node (Luan et al., 2022). Its diversification channel computes differences between node representations and their neighborhood aggregates. Unlike ACM, which constructs diversification within message-passing layers, FOCAL models changes across precomputed diffusion scales and learns their cross-hop fusion at the feature level.

3 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected attributed graph, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is the node set and $\mathcal{E}$ is the edge set. Its structure is represented by an adjacency matrix $A \in \{0,1\}^{n \times n}$. Specifically, $A_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$ and $A_{ij} = 0$ otherwise. The neighborhood of node v_i is defined as $\mathcal{N}_i = \{v_j \in \mathcal{V} \mid A_{ij} = 1\}$. The degree of v_i is $d_i = |\mathcal{N}_i| = \sum_j A_{ij}$, and the corresponding degree matrix is $D = \mathrm{diag}(d_1, \ldots, d_n)$. Each node v_i has a feature vector $x_i \in \mathbb{R}^d$. Stacking all node features gives $X = [x_1^\top; \ldots; x_n^\top] \in \mathbb{R}^{n \times d}$. Here, $X_{i:}$ is the feature vector of node v_i, while $X_{:f} \in \mathbb{R}^n$ contains the values of feature f across all nodes. Each node belongs to one of C classes, and $y_i \in \{1, \ldots, C\}$ denotes the label of node v_i.

We consider transductive node classification. The node set is divided into training, validation, and test sets, denoted by $\mathcal{V}_{\mathrm{tr}}$, $\mathcal{V}_{\mathrm{val}}$, and $\mathcal{V}_{\mathrm{te}}$. The graph structure and node features are available for all nodes. Training labels are used for parameter optimization, validation labels are used for model selection, and test labels are used only for evaluation.

3.1 Graph Convolution and Graph Diffusion

We define $\widetilde{A} = A + I_n$, where I_n is the identity matrix, and let $\widetilde{D}$ be its degree matrix. The normalized propagation operators without and with self-loops are $P_{\mathrm{n}} = D^{-1/2} A D^{-1/2}, P_{\mathrm{s}} = \widetilde{D}^{-1/2} \widetilde{A} \widetilde{D}^{-1/2}$. The n and s denote nonself-loop and self-loop propagation, respectively.

A graph convolutional layer combines neighborhood aggregation and feature transformation to update node representations as $Z^{(\ell+1)} = \mathrm{ReLU}\left(P_{\mathrm{s}} Z^{(\ell)} W^{(\ell)}\right)$, starting from $Z^{(0)} = X$ (Kipf & Welling, 2017). Here, $Z^{(\ell)}$ is the node representation at layer ℓ, and $W^{(\ell)}$ is a learnable parameter matrix. The multiplication by P_{s} performs neighborhood aggregation, while $W^{(\ell)}$ transforms the feature dimension.

Some decoupled GNNs separate graph propagation from feature transformation and compute propagated features before model training (Wu et al., 2019; Rossi et al., 2020). For $t \in \{\mathrm{n}, \mathrm{s}\}$, graph diffusion starts from the same input and repeatedly applies P_t:

$$H_t^{(0)} = X, \qquad H_t^{(k)} = P_t H_t^{(k-1)} = P_t^k X, \quad k = 1, \ldots, K. \tag{1}$$

We call $H_t^{(k)} \in \mathbb{R}^{n \times d}$ the Propagation State after k diffusion steps. The index k denotes the number of propagation steps.

ACM models neighborhood differences through the diversification operator $I_n - \widehat{A}$, where $\widehat{A}$ is a normalized neighborhood aggregation operator (Luan et al., 2022). Given a node representation R, the resulting diversification representation is $(I_n - \widehat{A})R = R - \widehat{A}R$. ACM uses this representation as a diversification channel together with its aggregation and identity channels within each GNN layer.

We use the same difference form to describe the change caused by each step of graph diffusion. For $t \in \{\mathrm{n}, \mathrm{s}\}$, the Propagation Change at step k is defined as

$$\Delta_t^{(k)} = H_t^{(k-1)} - H_t^{(k)} = (I_n - P_t) H_t^{(k-1)}, \quad k = 1, \ldots, K. \tag{2}$$

ACM uses the same algebraic difference form to construct a diversification channel within each neural layer. In FOCAL, $\Delta_t^{(k)}$ captures the feature differences between two consecutive precomputed Propagation States at the diffusion step k. Representing these changes explicitly allows FOCAL to learn their cross-hop weights independently from those of the Propagation States. The sequence $\{\Delta_t^{(1)}, \ldots, \Delta_t^{(K)}\}$ is therefore fused separately from the Propagation States in the following stage.

3.2 Cross-Hop Fusion

Neighborhood aggregation and cross-hop fusion describe different operations. Neighborhood aggregation combines information from connected nodes within one diffusion step. Cross-hop fusion combines the Propagation States produced at different diffusion steps. For a fixed propagation operator, we ignore the subscript t for simplicity. A common cross-hop fusion rule assigns a fixed or learned weight ω_k to each Propagation State and computes $U = \sum_{k=1}^{K} \omega_k H^{(k)} \in \mathbb{R}^{n \times d}$. Since ω_k is a scalar, the same hop weight is applied to every node and feature dimension.

Another choice concatenates all Propagation States into $U_{\mathrm{cat}} = H^{(1)} \,\|\, \cdots \,\|\, H^{(K)} \in \mathbb{R}^{n \times Kd}$, where $\|$ denotes concatenation. Weighted fusion and concatenation therefore use the same Propagation States but combine them in different forms. Building on these definitions, we then present FOCAL, which learns separate feature-wise weights for State and Change fusion.

4 Method

FOCAL learns from the Input, Propagation States, and Propagation Changes within a unified framework. As shown in Figure 2, graph diffusion first produces multi-hop Propagation States and Changes using the self-loop and nonself-loop operators. Feature-wise Fusion then combines them separately across propagation operators and diffusion steps to obtain the State representation U and Change representation V. The Input, State, and Change representations are transformed by three independent encoders. Node-level gates adjust the contributions of the encoded State and Change representations, while the encoded Input representation is directly used for prediction.

FOCAL performs Feature-wise Fusion over the combinations of propagation operators and diffusion steps. We introduce two parameter matrices $\Theta^u, \Theta^v \in \mathbb{R}^{d \times 2K}$, where each row corresponds to one feature dimension and each column corresponds to one combination of propagation operator and

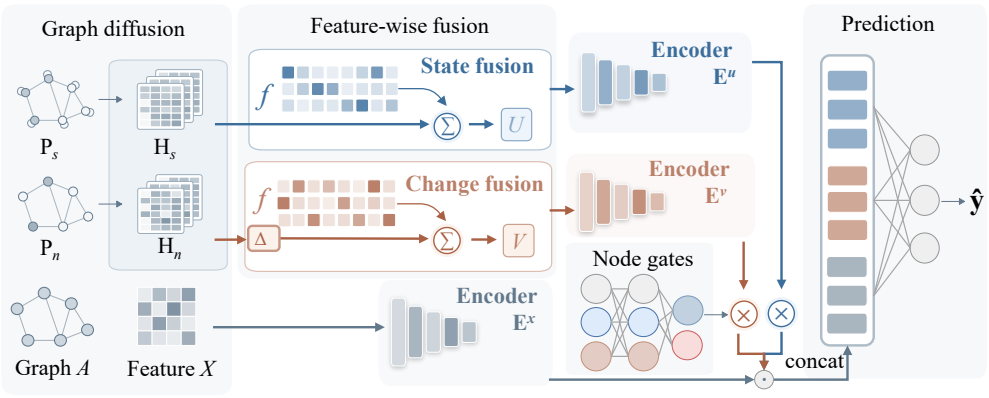


Figure 2: Overall framework of FOCAL. Multi-hop graph diffusion produces Propagation States and Changes, which are separately combined through Feature-wise Fusion. The fused State and Change representations are encoded together with the Input representation and integrated by Node gates for prediction.

diffusion step. For feature f, the State and Change fusion weights are computed as

$$\alpha_{f,t,k} = \frac{\exp\left(\theta^u_{f,t,k}/\tau - k\log 2\right)}{\sum_{t'\in\mathcal{T}}\sum_{\ell=1}^{K}\exp\left(\theta^u_{f,t',\ell}/\tau - \ell\log 2\right)}, \qquad \gamma_{f,t,k} = \frac{\exp\left(\theta^v_{f,t,k}/\tau - k\log 2\right)}{\sum_{t'\in\mathcal{T}}\sum_{\ell=1}^{K}\exp\left(\theta^v_{f,t',\ell}/\tau - \ell\log 2\right)}, \tag{3}$$

where $t \in \mathcal{T} = \{\mathrm{s}, \mathrm{n}\}$ and $k = 1, \ldots, K$. The temperature τ controls the distribution of the fusion weights. Both parameter matrices are initialized to zero. The term $-k\log 2$ gives the initial weights a decay proportional to 2^{-k}, assigning larger weights to shallower diffusion steps. For each feature, the weights are normalized over all propagation operators and diffusion steps.

Using these weights, the State and Change representations are obtained by combining the corresponding Propagation States and Changes

$$U_{if} = \sum_{t\in\mathcal{T}}\sum_{k=1}^{K}\alpha_{f,t,k}\left(H_t^{(k)}\right)_{if}, \qquad V_{if} = \sum_{t\in\mathcal{T}}\sum_{k=1}^{K}\gamma_{f,t,k}\left(\Delta_t^{(k)}\right)_{if}. \tag{4}$$

Both U and V belong to $\mathbb{R}^{n\times d}$. The weights $\alpha_{f,t,k}$ determine the contribution of each propagation scale to feature f in State fusion, while $\gamma_{f,t,k}$ determine its contribution in Change fusion. The two sets of parameters are learned independently, allowing the same feature to use different combinations of propagation operators and diffusion steps in the two branches.

After Feature-wise Fusion, the Input representation X, State representation U, and Change representation V are transformed into the same hidden space by three independent encoders. Let $R^x = X$, $R^u = U$, and $R^v = V$. The encoded representations are computed as

$$E^r = \mathrm{ReLU}\left(\mathrm{LayerNorm}\left(\mathrm{Dropout}(R^r)W^r + b^r\right)\right), \qquad r \in \{x, u, v\}, \tag{5}$$

where $W^r \in \mathbb{R}^{d\times h}$ and $E^r \in \mathbb{R}^{n\times h}$ for $r \in \{x, u, v\}$.

To adjust the contributions of the two graph-based representations at the node level, we concatenate the three encoded representations as $e_i = E^x_i \| E^u_i \| E^v_i$. The gate network takes e_i as input and produces two scalar gates

$$[g^u_i, g^v_i] = \sigma\left(\mathrm{ReLU}\left(e_i W^{g1} + b^{g1}\right)W^{g2} + b^{g2}\right), \tag{6}$$

where σ denotes the sigmoid function and $g_i^u, g_i^v \in (0,1)$. The two gates control the contributions of the encoded State and Change representations for node i. The representation used for prediction is $Z_i = E_i^x \| g_i^u E_i^u \| g_i^v E_i^v$. The encoded Input representation is kept unchanged by the gate network and is directly included in Z_i.

Stacking the representations of all nodes gives $Z \in \mathbb{R}^{n \times 3h}$. The final prediction is computed as $\hat{Y} = \mathrm{softmax}\left(\mathrm{Dropout}(Z)W^c + b^c\right)$, where $W^c \in \mathbb{R}^{3h \times C}$.

FOCAL jointly optimizes the parameters of Feature-wise Fusion, the three encoders, the Node gates, and the classifier. On the training node set $\mathcal{V}_{\mathrm{tr}}$, the model minimizes the cross-entropy loss $\mathcal{L} = -\frac{1}{|\mathcal{V}_{\mathrm{tr}}|} \sum_{v_i \in \mathcal{V}_{\mathrm{tr}}} \log \hat{Y}_{i,y_i}$.

During optimization, the precomputed Propagation States and Changes remain fixed, while U and V are updated according to the learned fusion weights. We use Adam to optimize all learnable parameters. Weight decay is applied to the encoders, Node gates, and classifier, while Θ^u and Θ^v are optimized without weight decay. The model with the highest validation accuracy is selected.

Complexity. Let m denote the total number of nonzero entries in the two propagation operators. The precomputation of the K-step Propagation States has the time complexity of $\mathcal{O}(Kmd)$. The per-epoch complexity of Feature-wise Fusion is $\mathcal{O}(Knd)$. Including the three encoders, Node gates, and classifier, the overall training complexity per epoch is $\mathcal{O}\big(Knd + ndh + nhh_g + nhC\big)$, where h_g denotes the hidden dimension of the gate network. The complete training procedure is summarized in Algorithm 1 in Appendix A.3.

5 Experiments

Our experiments are designed to answer the following questions.

RQ1. How does FOCAL compare with existing methods in node classification?

RQ2. Do propagation changes improve node classification?

RQ3. Does learning hop weights for each feature improve cross-hop fusion?

RQ4. Do State and Change benefit from separate fusion weights?

RQ5. How does sharing weights across both features and branches affect performance?

RQ6. Do node-level gates improve classification accuracy?

5.1 Experimental Setups

Datasets and evaluation. We evaluate FOCAL on eight node classification datasets, including Cora, CiteSeer, PubMed, WikiCS, Actor, Squirrel, Cornell, and Texas. The datasets cover different domains and levels of label homophily, with detailed statistics provided in Appendix A.1. For each dataset, we generate ten class-stratified random splits using 60%, 20%, and 20% of the nodes for training, validation, and testing, respectively. All methods are evaluated on the same splits. We report the mean test accuracy and standard deviation over the ten runs. All experiments follow the transductive setting, where the graph structure and node features are available for all nodes, while only training labels are used for optimization.

Baselines. We compare FOCAL with nine representative methods, including GAT (Veličković et al., 2018), SGC (Wu et al., 2019), APPNP (Gasteiger et al., 2019), GPR-GNN (Chien et al., 2021), and FSGNN (Maurya et al., 2021). We further include GFS (Zheng et al., 2025) and GumbelFS (Acharya & Zhang, 2020) for comparison with feature splitting and feature selection methods. ACM and ACMII (Luan et al., 2022) provide comparisons with node-level mixing of aggregation, diversification, and identity channels. We use their two-layer GCN implementations, denoted as ACM and ACMII.

Training settings. FOCAL precomputes Propagation States up to $K = 5$ using graph operators with and without self-loops. The three encoders use a hidden dimension of 128, while the hidden

Table 1: Test accuracy (%, mean ± standard deviation) over ten random splits. The highest mean accuracy in each column is shown in bold.

Method	Cora	CiteSeer	PubMed	WikiCS	Actor	Squirrel	Cornell	Texas
GAT	88.61 ± 1.61	75.24 ± 0.81	85.58 ± 0.36	80.73 ± 0.35	28.93 ± 0.46	27.77 ± 1.38	48.00 ± 3.13	58.86 ± 2.56
SGC	87.27 ± 1.11	74.55 ± 0.70	83.35 ± 0.64	80.84 ± 0.37	29.89 ± 0.63	34.75 ± 1.72	48.57 ± 2.86	58.29 ± 4.33
APPNP	87.98 ± 1.08	74.25 ± 1.87	87.34 ± 0.64	75.79 ± 0.80	31.45 ± 1.95	37.29 ± 1.74	46.29 ± 3.13	58.86 ± 2.56
GPR-GNN	$\mathbf{89.61 \pm 1.04}$	76.75 ± 1.06	88.13 ± 0.59	82.75 ± 0.49	32.88 ± 1.29	29.23 ± 1.43	56.57 ± 6.82	60.00 ± 2.02
FSGNN	86.27 ± 3.94	76.57 ± 1.36	89.61 ± 0.58	83.01 ± 0.64	36.19 ± 0.35	30.54 ± 1.55	78.29 ± 5.19	86.29 ± 6.19
GFS	85.01 ± 2.28	73.95 ± 1.49	87.73 ± 0.64	76.84 ± 1.19	35.25 ± 1.66	38.10 ± 1.21	60.57 ± 5.50	66.29 ± 7.11
GumbelFS	84.82 ± 1.36	73.92 ± 1.19	82.95 ± 0.69	78.33 ± 0.89	28.52 ± 1.18	34.52 ± 1.98	50.86 ± 8.43	60.57 ± 4.69
ACM	86.01 ± 2.49	75.36 ± 1.35	89.62 ± 0.63	81.10 ± 1.16	37.05 ± 1.01	33.73 ± 2.42	78.29 ± 4.78	85.14 ± 5.50
ACMII	86.38 ± 2.98	74.58 ± 1.22	89.46 ± 0.61	82.04 ± 0.65	36.19 ± 1.12	32.42 ± 2.43	76.57 ± 4.69	83.43 ± 4.69
FOCAL (Ours)	88.61 ± 2.15	$\mathbf{77.27 \pm 0.92}$	$\mathbf{90.76 \pm 0.55}$	$\mathbf{84.25 \pm 0.52}$	$\mathbf{37.10 \pm 0.89}$	$\mathbf{58.24 \pm 0.83}$	$\mathbf{83.71 \pm 4.27}$	$\mathbf{90.57 \pm 6.03}$

dimension of the gate network is 16. Input features are normalized row-wise using ℓ_1 normalization, and the dropout rate is set to 0.5. We train the model with Adam using a learning rate of 0.01. Weight decay is set to 5×10^{-4} for the encoder, gate, and classifier parameters and to zero for the Feature-wise Fusion parameters. The fusion temperature is set to 1. Training runs for at most 300 epochs with early stopping after 50 epochs without improvement. The same FOCAL configuration is used across all datasets. Additional implementation and training details are provided in Appendix A.2.

To ensure a fair comparison, all methods are evaluated using the same data splits, validation-based checkpoint selection, and training budget. For each baseline, we use the hyperparameter settings reported in the original paper.

Table 1 reports the node classification results. FOCAL achieves the highest mean accuracy on seven of the eight datasets. The largest improvements over the strongest baseline are observed on Squirrel, Cornell, and Texas. On Actor, FOCAL and ACM obtain similar mean accuracy. These results show consistent performance across datasets with different levels of label homophily.

GPR-GNN and FSGNN provide relevant comparisons for how models use multiple propagation scales. GPR-GNN learns one weight for each propagation step, shared across representation dimensions. It is the strongest baseline on Cora. FSGNN separately transforms precomputed multi-hop features, assigns a learned scalar weight to each feature matrix, and concatenates the outputs. It is among the strongest baselines on PubMed, Cornell, and Texas. FOCAL obtains higher mean accuracy than both methods on seven datasets. The design difference lies in how scale weights are assigned and where cross-hop fusion is performed. FOCAL learns scale weights for each input feature and fuses States and Changes separately before encoding. Each feature can therefore use different propagation scales for its State representation and its Change representation. This provides a direct way to learn which hops contribute to each feature in the two branches.

ACM and ACMII provide closely related comparisons because they also combine multiple graph channels through learned node-level weights. FOCAL obtains higher mean accuracy than both variants across all eight datasets, with the largest differences on WikiCS and Squirrel. The main distinction lies in how the graph representations are constructed before node-level weighting. ACM forms aggregation and diversification channels within message-passing layers, whereas FOCAL first performs feature-wise cross-hop fusion of precomputed Propagation States and Changes. Feature-wise weights determine the propagation scales used by each feature, while Node gates adjust the contributions of the encoded State and Change representations at the node level.

5.2 Ablation Studies

We evaluate five variants of FOCAL to address RQ2–RQ6. The variants modify the branch inputs, fusion weights, or Node gates, while all other experimental settings remain unchanged. Table 2 reports the results. Detailed definitions of the ablation variants are provided in Table 5 in Appendix A.3.

For **RQ2**, State-State replaces the Change branch with a second State branch while retaining separate fusion weights, encoders, Node gates, and the classifier. The parameter count therefore remains unchanged. State-State has lower accuracy than FOCAL across all eight datasets. Although its two branches can learn different scale weights, both operate on the same Propagation States. By

Table 2: Ablation results on the eight node classification datasets. Test accuracy is reported as mean ± standard deviation (%) over ten random splits.

Method	Cora	CiteSeer	PubMed	WikiCS	Actor	Squirrel	Cornell	Texas
Global-Fusion	87.05 ± 1.46	76.16 ± 0.95	90.44 ± 0.51	83.50 ± 0.77	35.99 ± 0.79	47.72 ± 7.74	81.14 ± 4.70	87.71 ± 5.05
Tied-Fusion	87.14 ± 1.28	76.39 ± 1.24	90.50 ± 0.60	83.80 ± 0.56	36.16 ± 0.71	49.18 ± 5.65	79.14 ± 3.58	87.14 ± 5.43
Global-Tied-Fusion	86.75 ± 1.61	76.40 ± 0.91	90.39 ± 0.51	83.38 ± 0.72	36.14 ± 1.12	46.01 ± 8.35	79.43 ± 3.24	85.14 ± 5.00
State-State	87.33 ± 1.56	76.39 ± 0.88	90.38 ± 0.63	83.85 ± 0.59	35.98 ± 1.09	42.90 ± 9.02	79.14 ± 4.68	85.14 ± 4.00
Fixed-Gates	86.88 ± 1.78	75.78 ± 1.26	90.53 ± 0.61	83.62 ± 0.75	36.15 ± 1.15	56.85 ± 1.19	78.86 ± 3.35	85.14 ± 4.63
FOCAL	$\mathbf{88.61 \pm 2.15}$	$\mathbf{77.27 \pm 0.92}$	$\mathbf{90.76 \pm 0.55}$	$\mathbf{84.23 \pm 0.67}$	$\mathbf{37.10 \pm 0.89}$	$\mathbf{58.24 \pm 0.83}$	$\mathbf{83.71 \pm 4.27}$	$\mathbf{90.57 \pm 6.03}$

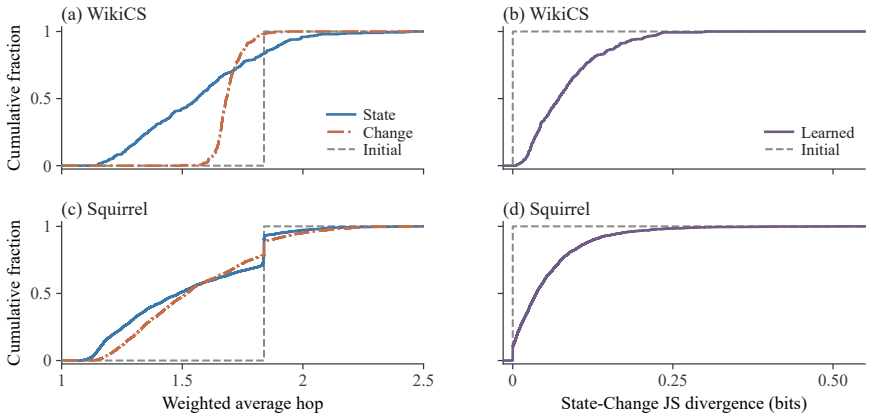


Figure 3: Learned scale weights on WikiCS and Squirrel. Subfigures (a,c) show the distributions of weighted average hop numbers. Subfigures (b,d) show the distributions of JS divergence between the State and Change weight vectors of the same feature. Gray dashed lines show the initial distributions.

contrast, the Change branch explicitly represents the differences between consecutive Propagation States, allowing changes introduced during graph diffusion to be fused separately from the states themselves.

To address **RQ3–RQ5**, we compare three variants that constrain the fusion weights in different ways. Global-Fusion uses one scale-weight vector for all feature dimensions within each branch, while retaining separate weights for State and Change. Tied-Fusion preserves feature-wise weights but shares them between the two branches. Global-Tied-Fusion applies both constraints and uses one scale-weight vector across all features and both branches. All three variants obtain lower mean accuracy than FOCAL across the evaluated datasets. The results support allowing fusion weights to vary across feature dimensions and between State and Change fusion.

For **RQ6**, Fixed-Gates removes the gate network and assigns both graph branches a fixed weight of 0.5 for every node. The Feature-wise Fusion parameters, three encoders, and classifier remain unchanged. Fixed-Gates obtains lower mean accuracy than FOCAL across all eight datasets. Feature-wise Fusion determines the propagation scales used by each feature, while Node gates further adjust the contributions of the encoded State and Change representations at the node level.

5.3 Analysis of Learned Scale Weights

To study how scale preferences vary across features and between State and Change, we analyze the learned fusion weights on WikiCS and Squirrel with $K = 5$. Each feature has 10 normalized weights over two propagation operators and five diffusion steps in each branch. We measure the scale preference of feature f by its weighted average hop, computed as $\sum_{t\in\mathcal{T}} \sum_{k=1}^{K} k\alpha_{f,t,k}$ for State and $\sum_{t\in\mathcal{T}} \sum_{k=1}^{K} k\gamma_{f,t,k}$ for Change. Since all features and both branches share the same initialization, differences in learned distributions reflect scale preferences across features and branches.

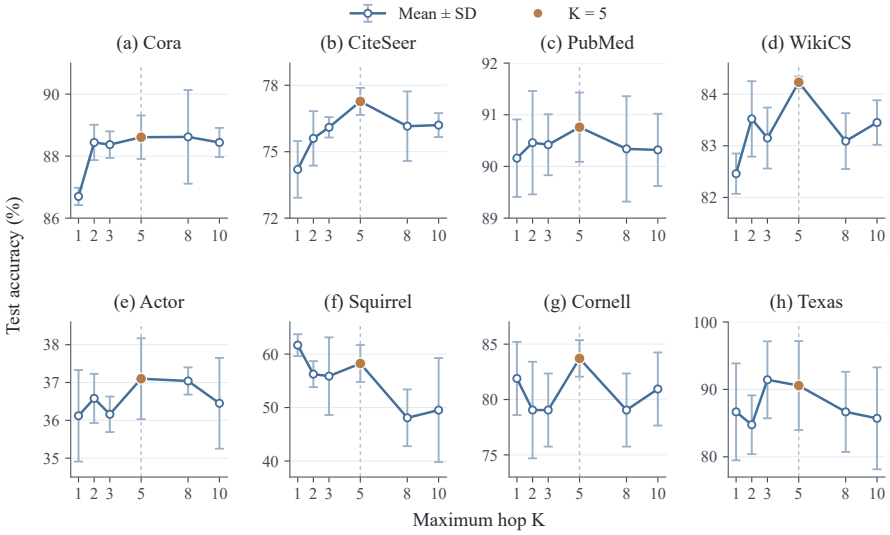


Figure 4: Test accuracy of FOCAL for different maximum diffusion hops K across eight datasets. Filled markers and dashed lines indicate $K = 5$.

Figure 3 (a,c) show the distributions of weighted average hops. On WikiCS, State exhibits a wider range of learned hop preferences than Change. On Squirrel, both branches also show variation across features, although their overall distributions are similar. These patterns indicate that different features learn different propagation scales rather than following a common hop preference.

We further compare the complete State and Change weight vectors of each feature using Jensen–Shannon divergence (Menéndez et al., 1997). Figure 3 (b,d) show mean divergences of 0.084 and 0.054 bits on WikiCS and Squirrel, respectively. In particular, similar weighted average hops on Squirrel do not imply identical fusion weights, as different weight distributions can produce the same average hop. Features can assign different weights to propagation operators and diffusion steps while maintaining similar average hop values. The learned weights therefore capture differences across both features and the State and Change branches.

5.4 Sensitivity to Diffusion Depth

Sensitivity to K. We evaluate the effect of diffusion depth by varying the maximum hop $K \in \{1, 2, 3, 5, 8, 10\}$ while keeping all other settings fixed. Figure 4 shows that the preferred diffusion depth varies across datasets. Cora, CiteSeer, WikiCS, and Cornell generally benefit from increasing K to a moderate value, whereas PubMed remains stable across different values of K. Actor shows a non-monotonic trend and reaches its highest mean accuracy at $K = 5$. Squirrel performs best at $K = 1$, while Texas reaches its highest accuracy at $K = 3$. Across all eight datasets, $K = 5$ achieves the highest mean accuracy on five datasets and remains close to the best result on Cora. We therefore use $K = 5$ in all experiments.

6 Conclusion

We presented FOCAL, a graph learning framework that models Propagation States and Changes through feature-wise cross-hop fusion. FOCAL learns independent scale weights for each feature in the State and Change branches and further uses Node gates to adjust their contributions at the node level. Experiments show that FOCAL achieves the highest mean accuracy on seven of the eight datasets. The ablation studies further demonstrate the contributions of Propagation Changes, feature-wise fusion, separate State and Change weights, and Node gates. Analysis of the learned weights shows that scale preferences vary across features and between the two branches, supporting separate modeling of feature values after diffusion and their changes across diffusion steps.

AI USE STATEMENT

Generative AI tools were used solely for language editing and polishing of the manuscript. They were not used for research design, methodology development, experiments, data analysis, or scientific conclusions. All AI-assisted edits were reviewed and revised by the authors. The authors take full responsibility for the final content of the manuscript.

REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, volume 97, pp. 21–29, 2019.

Deepak Bhaskar Acharya and Huaming Zhang. Feature selection and extraction for graph neural networks. In *ACM Southeast Conference*, pp. 252–255, 2020. doi: 10.1145/3374135.3385309.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 3950–3957, 2021.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(1), 2023.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

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

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. Proceedings of the AAAI Conference on Artificial Intelligence, 2018.

Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. *Advances in Neural Information Processing Systems*, 35:1362–1375, 2022.

Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Improving graph neural networks with simple architecture design. *ArXiv*, 2021.

María Luisa Menéndez, Julio Angel Pardo, Leandro Pardo, and María del C Pardo. The jensen-shannon divergence. *Journal of the Franklin Institute*, 334(2):307–318, 1997.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

Emanuele Rossi, Fabrizio Frasca, Benjamin Paul Chamberlain, Davide Eynard, Michael M. Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *ArXiv*, abs/2004.11198, 2020.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

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

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, volume 97, pp. 6861–6871, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, volume 80, pp. 5453–5462, 2018.

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

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *International Conference on Neural Information Processing Systems*, pp. 4805–4815, 2018.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *International Conference on Neural Information Processing Systems*, pp. 5171–5181, Red Hook, NY, USA, 2018. Curran Associates Inc.

Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: a theory of using graph neural networks for multi-node representation learning. In *International Conference on Neural Information Processing Systems*, 2021.

Yilun Zheng, Xiang Li, Sitao Luan, Xiaojiang Peng, and Lihui Chen. Let your features tell the differences: Understanding graph convolution by feature splitting. In *International Conference on Learning Representations*, 2025.

A Appendix

A.1 Dataset Details

Table 3: Statistics of the eight node classification datasets.

Dataset	Type	#Nodes	#Edges	#Features	#Classes	Homophily
Cora	Citation	2,708	5,278	1,433	7	0.81
CiteSeer	Citation	3,327	4,552	3,703	6	0.74
PubMed	Citation	19,717	44,324	500	3	0.80
WikiCS	Wikipedia	11,701	216,123	300	10	0.65
Actor	Actor co-occurrence	7,600	26,752	932	5	0.22
Squirrel	Wikipedia	5,201	198,493	2,089	5	0.22
Cornell	WebKB	183	280	1,703	5	0.30
Texas	WebKB	183	295	1,703	5	0.11

Homophily. Table 3 reports the edge homophily ratio h_{edge} for each dataset following the definition of Pei et al. (2020). Given a graph with edge set $\mathcal{E}$ and node labels y, edge homophily is defined as the proportion of edges whose endpoints share the same label,

$$h_{\text{edge}} = \frac{1}{|\mathcal{E}|} \sum_{(v_i, v_j) \in \mathcal{E}} \mathbb{1}[y_i = y_j]. \tag{7}$$

A larger h_{edge} indicates that connected nodes are more likely to share the same label, while a smaller h_{edge} indicates that they are more likely to have different labels. Accordingly, Cora, CiteSeer, PubMed, and WikiCS are treated as homophilic datasets, while Actor, Squirrel, Cornell, and Texas are treated as heterophilic datasets in our experiments.

A.2 Implementation Details

Before graph diffusion, each input row is normalized as $X_{i:} \leftarrow X_{i:} / \max(1, \|X_{i:}\|_1)$. The normalized feature matrix is used by the Input branch and both diffusion operators. The self-loop operator is constructed from $A + I_n$, where A denotes the adjacency matrix, while the nonself-loop operator is constructed after removing the diagonal elements of A. Each operator applies its own degree normalization. No additional modification is applied to the edge structure. If the stored graph already contains self-loops, these elements remain in the self-loop branch when I_n is added.

For each operator $t \in \{s, n\}$, we set $H_t^{(0)} = X$ and precompute $H_t^{(k)} = P_t H_t^{(k-1)}$ for $k = 1, \ldots, K$. These Propagation States remain fixed during training. The implementation stores X together with the $2K$ Propagation States. Change fusion uses the differences $H_t^{(k-1)} - H_t^{(k)}$ computed from the stored Propagation States.

State and Change fusion use separate trainable logit matrices $\Theta^u, \Theta^v \in \mathbb{R}^{d \times 2K}$. For each feature, softmax normalization is applied jointly across both propagation operators and all diffusion hops. The State and Change fusion weights are computed as $\alpha_{f,t,k} = \mathrm{softmax}_{(t,k)}(\theta^u_{f,t,k}/\tau - k \log 2)$ and $\gamma_{f,t,k} = \mathrm{softmax}_{(t,k)}(\theta^v_{f,t,k}/\tau - k \log 2)$, respectively. Both trainable logit matrices are initialized to zero, and we set $\tau = 1$. The fixed bias $-k \log 2$ gives initial weights proportional to 2^{-k}, with equal initial weights assigned to the two operators at the same hop.

The Input, State, and Change branches use separate encoders, each consisting of a linear transformation from d to h dimensions followed by LayerNorm and ReLU. Dropout is applied to each branch representation before the encoder. The gate network takes the concatenated representation $[E_i^x \| E_i^u \| E_i^v]$ as input and applies a linear transformation from $3h$ to 16 dimensions, followed by ReLU and a second linear transformation from 16 to two dimensions. Sigmoid activation is then applied to obtain g_i^u and g_i^v. The final gate layer is initialized to zero, giving $g_i^u = g_i^v = 0.5$ at initialization. The two gates are learned independently and are not constrained to sum to one. The classifier takes $[E_i^x \| g_i^u E_i^u \| g_i^v E_i^v]$ as input, followed by dropout and a linear classification layer.

Table 4: FOCAL training configuration.

Parameter	Value
Maximum diffusion hop K	5
Encoder hidden dimension h	128
Gate hidden dimension	16
Dropout probability	0.5
Softmax temperature τ	1.0
Optimizer	Adam
Learning rate for fusion logits	0.01
Learning rate for other parameters	0.01
Weight decay on fusion logits	0
Weight decay on other parameters	5×10^{-4}
Maximum training epochs	300
Early-stopping patience	50

Algorithm 1 Training FOCAL

Require: Graph A, features X, labels y, saved split masks, maximum hop K
Ensure: Trained model and test accuracy

Normalize X and construct P_{s} and P_{n}
Set $H_s^{(0)} = H_n^{(0)} = X$
for $k = 1, \ldots, K$ **do**
    Compute $H_{\mathrm{s}}^{(k)} = P_{\mathrm{s}} H_{\mathrm{s}}^{(k-1)}$ and $H_{\mathrm{n}}^{(k)} = P_{\mathrm{n}} H_{\mathrm{n}}^{(k-1)}$
end for
Initialize fusion logits, encoders, gate network, and classifier
Initialize Adam and the checkpoint record
for each training epoch **do**
    Compute α and γ by softmax over the $2K$ operator-hop combinations
    Compute $U_{:f} = \sum_{t,k} \alpha_{f,t,k} H_{t,:f}^{(k)}$ for each feature f
    Compute $V_{:f} = \sum_{t,k} \gamma_{f,t,k} (H_{t,:f}^{(k-1)} - H_{t,:f}^{(k)})$ for each feature f
    Encode X, U, and V to obtain E^x, E^u, and E^v
    Compute (g_i^u, g_i^v) from $[E_i^x \| E_i^u \| E_i^v]$ for each node i
    Classify $[E_i^x \| g_i^u E_i^u \| g_i^v E_i^v]$
    Update all trainable parameters using training cross-entropy
    Evaluate validation accuracy and cross-entropy
    Save the checkpoint if the validation selection rule improves
    Stop if the checkpoint has not improved for 50 epochs
end for
Restore the saved checkpoint
Evaluate test accuracy with dropout disabled

We use full-batch training with one optimizer update per epoch and optimize the classification cross-entropy over the training nodes. Adam applies weight decay to the encoder, gate, and classifier parameters, while the fusion logits are optimized without weight decay. After each epoch, validation accuracy and cross-entropy are evaluated with dropout disabled. A checkpoint is saved when validation accuracy improves, or when the same accuracy is obtained with lower validation cross-entropy. Training stops after 50 epochs without checkpoint improvement or after reaching 300 epochs. The saved checkpoint is then restored for test evaluation without further training.

A.3 Algorithm and Ablation Details

Algorithm 1 summarizes the training procedure of FOCAL. Graph diffusion is performed before optimization, after which the fusion logits, three encoders, gate network, and classifier are jointly optimized using the classification objective. Validation accuracy is used for checkpoint selection.

Table 5: Ablation definitions. The "Across features" and "Across branches" columns indicate whether fusion weights are shared among features and between the State and Change branches, respectively.

Variant	Across features	Across branches	Second branch	Gates
FOCAL	Separate	Separate	Change	Learned
State-State	Separate	Separate	State	Learned
Global-Fusion	Shared	Separate	Change	Learned
Tied-Fusion	Separate	Shared	Change	Learned
Global-Tied-Fusion	Shared	Shared	Change	Learned
Fixed-Gates	Separate	Separate	Change	Fixed

Table 5 summarizes the modifications introduced by each ablation variant. All variants use the same Propagation States, Input branch, encoder architectures, classifier width, training objective, and checkpoint selection rule. Each variant is trained separately with a new initialization.

State-State replaces each Propagation Change in the second graph branch with the corresponding Propagation State. The two graph branches retain separate fusion weights and encoders, and the parameter count remains the same as that of FOCAL. Global-Fusion uses one $2K$-dimensional logit vector for each branch, shared across all feature dimensions. Tied-Fusion uses a single $d \times 2K$ trainable logit matrix shared by State and Change, such that both branches use the same fusion parameters. Global-Tied-Fusion combines these two constraints and uses a single $2K$-dimensional logit vector shared across all features and both branches. Fixed-Gates removes the gate network and sets $g_i^u = g_i^v = 0.5$ for every node while keeping the fusion weights and encoders trainable.