LOOPING IN CONTEXT: NODE CLASSIFICATION WITH LOOPED PRIOR-FITTED NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Prior-fitted networks (PFNs) for graphs classify nodes on arbitrary graphs in a single forward pass. However, how their predictions are formed across layers remains unclear. For tabular foundation models, mechanistic studies show that depth performs iterative *refinement* and that a single looped transformer block can recover a full-depth model. We investigate whether this holds for graphs, where each layer also performs one hop of *propagation*. Six mechanistic experiments on the 12-layer `NodePFN` across 23 benchmarks show that depth plays two roles. The attention branch transfers the labels in the first layer and becomes redundant and self-repairing in later layers. The message-passing branch remains necessary on homophilous graphs, where the loss caused by removing one hop persists to the last layer. Consequently, the depth at which predictions form increases with homophily in both `NodePFN` and `GraphPFN`, and a two-class analysis explains why no fixed depth is optimal across homophily levels. We therefore propose **Looped NodePFN**, which applies a single block repeatedly and selects its depth for each graph *in context* from held-out labeled nodes, without gradient updates or additional parameters. With one-tenth of the parameters, it achieves 66.3% mean accuracy, compared with 66.6% for the 12-layer model, and predicts up to 5.1 times faster once its depth is selected. Thus, a single block suffices for PFNs for graphs, provided that its iterations and message-passing steps are chosen per graph.

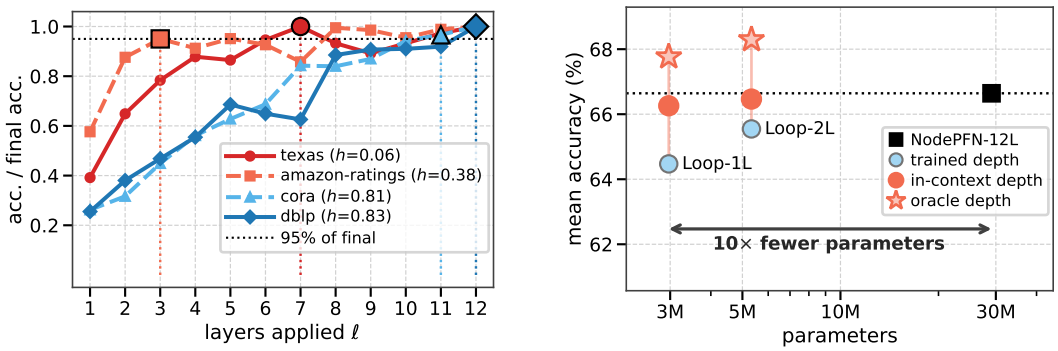
1 INTRODUCTION

Prior-fitted networks (PFNs) for graphs classify the nodes of an arbitrary graph in a single forward pass, taking the labeled nodes as context and predicting the unlabeled ones with no dataset-specific training (Choi et al., 2026; Eremeev et al., 2026), in the manner of prior-fitted networks for tabular data (Müller et al., 2022; Hollmann et al., 2023; 2025). Two such models have been proposed in the past year. `NodePFN` (Choi et al., 2026) stacks 12 layers, each combining attention over the context with message passing over the graph (29M parameters, `NodePFN-12L` below), and `GraphPFN` (Eremeev et al., 2026) wraps a frozen tabular foundation model with a graph adapter after every layer. These models apply the same depth to every graph, whether or not neighboring nodes share labels. This depth determines the number of parameters and the computational cost per query. However, it remains unclear how such models use their depth, which raises two practical questions. How small can a PFN for graphs be, and can its computation be adapted to the input graph?

The first question was recently studied for tabular foundation models. Balef et al. (2026) showed that depth in a TabPFN-style model performs iterative refinement, that middle layers are largely redundant, and that one transformer block looped six times recovers a six-layer model. Their study does not address *when* more depth is needed or how a frozen model could choose it at test time.

Hypothesis: one block may suffice if its depth depends on the graph. For graphs the first question cannot be separated from the second, because each layer of a PFN for graphs performs two operations. An attention branch refines the query representations against the labeled context, and a message-passing branch expands every node's receptive field by one hop. As a result, depth determines both the amount of refinement (*how long to think*) and the size of the receptive field (*how far to look*), and a stack of layers couples the two. The appropriate receptive field, however, depends on the graph. Let h be the edge homophily, the fraction of edges joining same-label nodes (Zhu

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.



(a) Prediction depth varies by graph

(b) Accuracy vs. parameters

Figure 1: **Each graph needs its own depth, and one looped block can provide it.** (a) NodePFN without feature pre-smoothing (k=0, §3.2), large markers at the first layer within 95% of final accuracy. Graphs with high h are homophilous, those with low h heterophilous. (b) ICDS lifts Loop-1L from 64.5% to 66.3%, against 66.6% for NodePFN-12L with ten times the parameters.

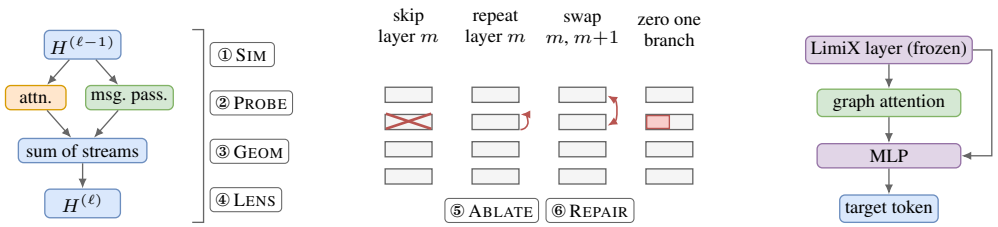
et al., 2020). Propagation is beneficial when h is high and smooths out the class signal when h is low (Oono & Suzuki, 2020; Rusch et al., 2023; Ma et al., 2022). A GNN trained on the target graph can tune its depth to h. A PFN for graphs is pretrained once and never trained on the target graph, so its depth is either fixed for every graph or chosen at inference. Therefore, if one block is sufficient, its number of iterations is expected to increase with h. Figure 1a supports this for NodePFN, whose predictions are complete by layer 7.3 on heterophilous and by layer 10.2 on homophilous graphs, on average over 19 benchmarks (§3.2).

Findings: one block suffices, and the depth depends on the graph. We test this hypothesis with six mechanistic experiments on NodePFN that separate the two branches and repeat them on GraphPFN (§3). The two roles are not symmetric. The attention branch transfers the labels in the first layer and only refines in later layers. When a later attention block is removed, the layers that follow repair the loss, so one block can replace them. The message-passing branch adds one hop per layer, and later layers only partly repair the loss caused by removing a hop. Consequently, the depth required by a graph increases with its homophily in both NodePFN and GraphPFN.

Method: loop one block, and pick the depth per graph. A two-class analysis further shows that no single depth is optimal for all graphs. To this end, we propose *looping in context* and build Looped NodePFN on it (§4). One NodePFN block, with both its attention and message-passing branches, replaces the interchangeable layers, and the labeled context determines how many times the block is applied. We train the block to be looped and, at inference, score each iteration *in context* on a held-out subset of the labeled nodes and select the best iteration, with no gradient step and no extra parameter (in-context depth selection, ICDS). In Figure 1b the looped block reaches 66.3% against 66.6% for NodePFN-12L with a tenth of the parameters, and its depth follows the graph. An oracle depth would increase its accuracy to 67.8%, which leaves room for better depth selection. These results answer both questions, since a PFN for graphs can consist of a single block whose computation is adapted to each graph.

Our contributions are:

1. **A branch-level mechanistic study of NodePFN, reproduced on GraphPFN** (§3). Labels are transferred once, by the first layer's attention, and later attention is self-repairing. Message passing is needed only on homophilous graphs, so prediction depth grows with homophily.
2. **Looped NodePFN with in-context depth selection** (§4). A contextual stochastic block model (cSBM) analysis shows that no fixed depth is optimal across homophily levels (Proposition 1), so one NodePFN block is looped and ICDS picks its iterations per graph from the labeled context.
3. 10× **fewer parameters, faster prediction, no loss in accuracy** (§5). Looped NodePFN with a one-layer block (2.97M parameters vs. 29.0M) reaches NodePFN-12L's mean accuracy (Fig. 1b, Table 1) and, because ICDS selects fewer iterations than the training depth on 19 of the 23 graphs, predicts up to 5.1 times faster (2.2 times on the median graph, Appendix E). An oracle depth raises it to 67.8%, and the chosen depth follows h on the benchmarks and cSBM.



(a) Readouts on a NodePFN layer (b) Interventions on the execution order (c) GraphPFN layer

Figure 2: **What we measure, and where.** (a) A NodePFN layer runs attention (over context and query rows) and graph message passing in parallel and sums them, and the residual stream after every layer is read by ① SIM, ② PROBE, ③ GEOM and ④ LENS (§3.1–3.2). (b) Interventions on the execution order (skip, repeat or swap a layer, or zero one branch in red), scored by ⑤ ABLATE and ⑥ REPAIR (§3.3). (c) A GraphPFN layer (a frozen LimiX layer, a graph-attention adapter and an MLP), whose target token is read in the same way (Appendix C).

2 BACKGROUND

NodePFN. NodePFN (Choi et al., 2026) is pretrained on synthetic graphs whose homophily spans $h \in [0.1, 0.9]$. Nodes form a sequence $[\,\text{context}; \text{query}\,]$, in which context rows carry their label in the input embedding and query rows do not. Each of its $L = 12$ GPS-style layers (Rampášek et al., 2022) maintains a residual stream $H^{(\ell)} \in \mathbb{R}^{N\times d}$ and computes

$$\begin{aligned} H_{\text{mp}} &= \text{LN}_{\text{mp}}\big(H^{(\ell)} + \text{GCN}(H^{(\ell)}, A)\big), & H_{\text{att}} &= \text{LN}_{\text{att}}\big(H^{(\ell)} + \text{Attn}(H^{(\ell)})\big), \\ H^{(\ell+1)} &= \text{LN}_2\big(\tilde{H} + \text{FFN}(\tilde{H})\big), & \tilde{H} &= H_{\text{mp}} + H_{\text{att}}, \end{aligned} \tag{1}$$

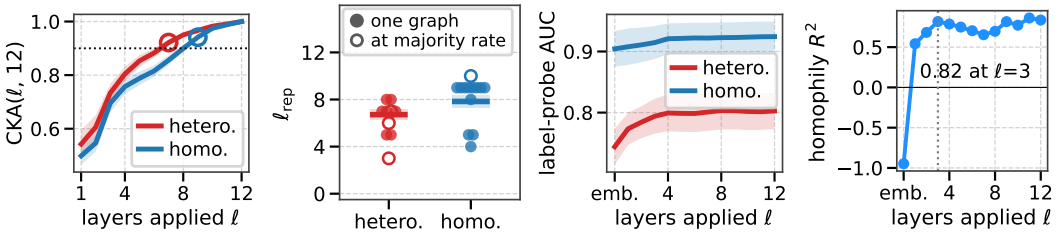
where GCN is a one-hop graph convolution over the normalized adjacency (with self-loops) and Attn applies self-attention among context rows and cross-attention from query rows to context rows, as in PFNs. The two branches are computed in parallel from the same input and *summed* as two separately normalized residual streams, which allows us to intervene on one branch while leaving the other intact (Fig. 2a). NodePFN-12L has $d = 512$, 4 heads, and 29.0M parameters.

GraphPFN. GraphPFN (Eremeev et al., 2026) wraps every layer of a frozen tabular foundation model (LimiX) with a zero-initialized graph-attention adapter and an MLP (Fig. 2c). Only the adapters were trained, and its own evaluation pipeline applies no feature smoothing. We use it as a second, architecturally different PFN for graphs on which to repeat the depth readouts, with details in Appendix C. Other in-context graph models (Liu et al., 2024; Tang et al., 2024; Zhao et al., 2025; Platonov et al., 2026) and tabular PFNs (Hollmann et al., 2023; 2025; Qu et al., 2025) are related but outside the scope of this study.

Analysis protocol. We reuse the mechanistic protocol that Balef et al. (2026) developed for tabular foundation models wherever possible, so that differences we report are attributable to the graph setting rather than to methodology. Its readouts on the residual stream are embedding similarity, probing, separation gap and a logit lens with per-layer tuned decoders (Fig. 2a), and its interventions are layer skip, repeat and swap with a self-repair readout (Fig. 2b). Because the two branches of a NodePFN layer are summed, we can also zero one branch of one layer while leaving the other intact (red in Fig. 2b). Section 6 contrasts their findings with ours.

Depth and homophily. Repeated propagation drives node representations toward a degree-weighted constant, which the Dirichlet energy $E(H) = \text{tr}(H^\top \tilde{L} H)/\|H\|_F^2$ measures (Oono & Suzuki, 2020; Cai & Wang, 2020; Rusch et al., 2023), and whether propagation improves or degrades linear separability depends on homophily and degree (Baranwal et al., 2021; Ma et al., 2022; Luan et al., 2023). Both quantities recur in §3 and §4.1.

Notation. $H^{(\ell)}$ is the residual stream after ℓ layers, ρ is Spearman's rank correlation across graphs (over the 19 informative ones unless stated), accuracy is reported in %, and methods are compared with a paired Wilcoxon signed-rank test over graph–split pairs.



(a) CKA to the final layer (b) ℓ_{rep} by group (c) label probe (d) homophily probe

Figure 3: ① SIM **Representations settle earlier on heterophilous graphs** (a, b). ② PROBE **Labels are transferred once and the graph is diagnosed by layer 3** (c, d). Group mean ± s.e. Red: heterophilous, blue: homophilous, here and in every later figure. (a) Circles at ℓ_{rep}, the first layer with CKA above 0.9. (b) One dot per graph, and ℓ_{rep} rises with h. (d) Leave-one-graph-out R^2.

3 WHAT DEPTH DOES IN A PFN FOR GRAPHS

We use two kinds of experiments to examine the role of depth (Fig. 2). Readouts examine the unchanged model after each layer to determine what the representation encodes (§3.1) and when the prediction is ready (§3.2). Interventions alter single layers to find which computations are needed (§3.3). Every analysis separates homophilous from heterophilous graphs and is repeated on `GraphPFN` (Appendix C).

Setup. We analyze the released 12-layer `NodePFN` (`NodePFN-12L`) in a single forward pass of the official inference code (Appendix A), without ensembling, so that each readout reflects a single computation. That pipeline smooths the input features k times ($X \leftarrow \hat{A}X$) before the model, with a k fixed per dataset. We call this pipeline with `NodePFN`'s own k the *standard* protocol, and the same pipeline with k=0 for every graph the *no-smoothing* condition (§3.2). The benchmarks are the 23 graphs of Choi et al. (2026), 13 homophilous and 10 heterophilous. Four binary graphs at majority-class accuracy are drawn hollow and excluded from correlations. Early-exit readouts use per-layer decoders trained on `NodePFN`'s prior with a frozen backbone (Belrose et al., 2023; Appendix A).

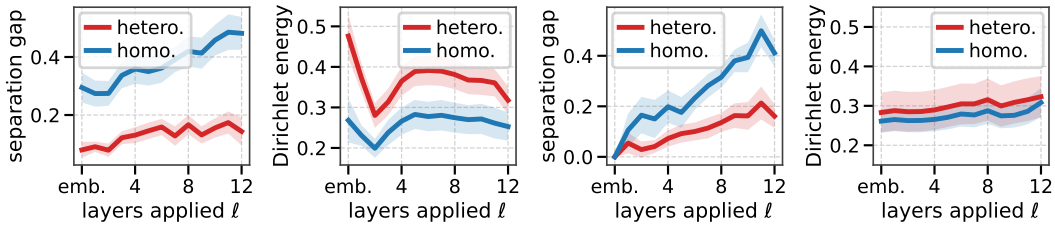
3.1 REPRESENTATION ANALYSIS

① **SIM Embedding similarity.** Balef et al. (2026) found that most layers of a tabular foundation model barely change its representation. We first examine whether this also holds for graphs. Figure 3a shows linear CKA (Kornblith et al., 2019) between each layer's representation and the final one. A CKA near one means that two layers encode the same geometry, so a block of high CKA marks layers that refine the representation instead of restructuring it. In both groups the input embedding is isolated and layers ≥ 4 form one block, as in tabular models. On heterophilous graphs the block starts about two layers earlier (ℓ_{rep} of 6.3 vs. 8.0, Fig. 3b), which suggests, depending on the CKA threshold (Appendix A), that the graph affects how much depth the model uses.

Takeaway 1. Most layers only refine the representation, as in tabular models, which makes one looped block plausible. On heterophilous graphs this stage starts two layers earlier.

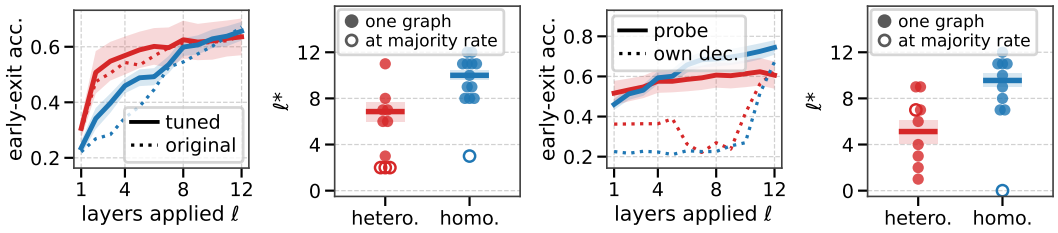
② **PROBE Label and homophily probes.** Similarity does not reveal what the representation contains, so we probe it. A linear probe for the query labels asks when the information needed to classify the queries is present. Its accuracy is at ceiling from layer 1 onward (Fig. 3c), so this information reaches the queries in the first layer and later layers do not add it. A second probe tests whether the model encodes the type of graph, to which its depth would have to adapt. A ridge probe for homophily on the pooled query embedding, evaluated leave-one-graph-out, explains none of the variance of h at the input and 82% of it after three layers, with no further gain (Fig. 3d).

Takeaway 2. The first layer carries the labels to the queries, and by layer 3 the model encodes the graph's homophily, long before its last layer.



(a) NodePFN: separation (b) NodePFN: energy (c) GraphPFN: separation (d) GraphPFN: energy

Figure 4: ③ GEOM **Separation grows with depth, and smoothing happens early.** Separation gap Δ_ℓ and Dirichlet energy E per layer, group mean $\pm$ s.e. (GraphPFN: target-token states).



(a) NodePFN: accuracy (b) NodePFN: ℓ^* (c) GraphPFN: accuracy (d) GraphPFN: ℓ^*

Figure 5: ④ LENS **Without pre-smoothing, the prediction is ready at a deeper layer on homophilous graphs.** NodePFN without pre-smoothing (k=0), GraphPFN under its own protocol. Solid: tuned decoders (NodePFN) or a linear probe (GraphPFN). Dotted: the model's own decoder. ℓ^*: first layer within 95% of final accuracy. Standard protocol in Fig. 10.

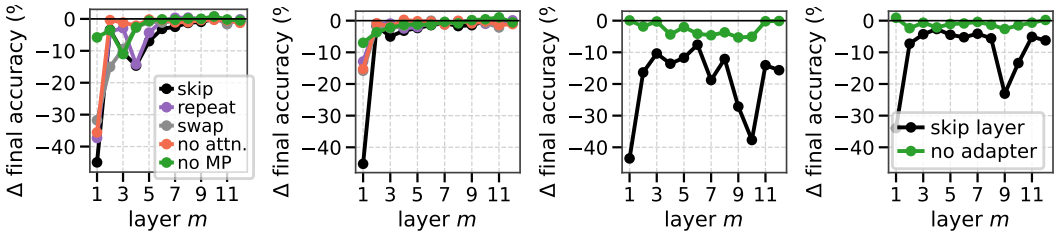
③ GEOM Separation gap and Dirichlet energy. Probes show what information is present. The next two readouts show how the classes are arranged and whether propagation makes them less distinguishable. Following Balef et al. (2026) we compute the separation gap Δ_ℓ between and within classes, and add the Dirichlet energy E (Fig. 4). A drop in E means that neighboring nodes have become more alike, so it marks where propagation smooths the representation. Δ_ℓ grows with depth in both groups, so every layer keeps separating the classes. The energy drops within the first two layers on heterophilous graphs ($0.60 \rightarrow 0.30$) and then remains constant, and it is flat on homophilous graphs, whose features already match the graph. GraphPFN shows the same growth without the early dip. Deeper layers therefore do not smooth the representation further.

Takeaway 3. Classes separate further with every layer. Message passing smooths the representation only in the first two layers and only on heterophilous graphs, not in deeper ones.

3.2 A GRAPH LOGIT LENS

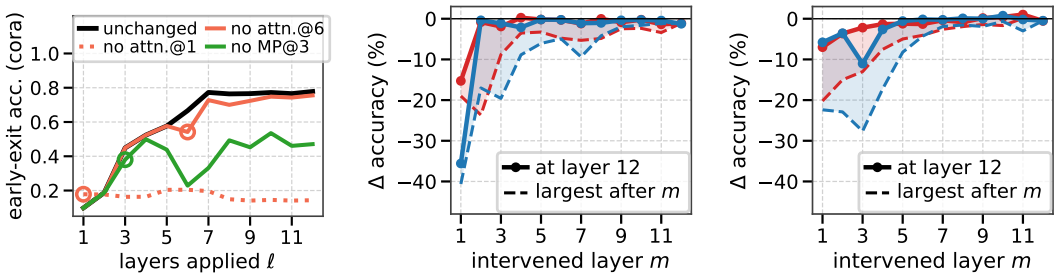
④ LENS Graph logit lens. The readouts so far describe representations. The lens measures when the prediction itself is ready, which is the quantity that a depth rule must choose. With the tuned decoders we measure early-exit accuracy at every layer and define $\ell^*(G)$ as the first layer within 95% of the final value (Fig. 5). When the model performs all of the propagation, ℓ^* increases with homophily, from 7.3 layers on heterophilous graphs to 10.2 on homophilous ones, and its rank order across graphs follows h. GraphPFN, whose pipeline never smooths, shows the same. Under NodePFN's standard protocol ℓ^* no longer varies with h, because the tuned pre-smoothing already propagates homophilous graphs by two to four hops. Removing it shifts only the homophilous group, by about as many layers as smoothing steps were removed (Appendix B, Fig. 10). The required depth is therefore a property of the graph, and part of it can be moved into preprocessing.

Takeaway 4. In both models the prediction is ready earlier on heterophilous graphs than on homophilous ones (7.3 vs. 10.2 layers in NodePFN). NodePFN's standard protocol hides this, because its pre-smoothing propagates before the model does.



(a) NodePFN, homo. (b) NodePFN, hetero. (c) GraphPFN, homo. (d) GraphPFN, hetero.

Figure 6: ⑤ ABLATE **Only layer 1 is essential, and only its attention.** Change in final accuracy (%) when layer m is skipped (black), repeated (purple), swapped (gray), or has one branch zeroed (coral: attention, green: message passing or adapter). GraphPFN: skip and adapter only.



(a) Cora: one branch zeroed (b) attention zeroed at m (c) message passing zeroed at m

Figure 7: ⑥ REPAIR **Later layers undo attention removals, but not early message-passing removals on homophilous graphs.** (a) Early-exit accuracy of the unchanged model (black) and with one branch of one layer zeroed (circle: intervened layer). (b, c) Largest drop at any later layer (dashed) and drop at layer 12 (solid), group means. Shading: what later layers repaired.

3.3 BRANCH-WISE LAYER ABLATION AND SELF-REPAIR

⑤ ABLATE Layer and branch ablation. Readouts cannot determine which computations the model needs, so we modify the execution without changing the weights. We skip, repeat or swap layers, or zero one branch of one layer, and measure final accuracy (Fig. 6). Skipping tests whether a layer is needed at all, and zeroing a branch tests whether the layer needs its attention or its message passing. Skipping layer 1 lowers final accuracy from 69% and 64% to about 20% in the two groups, and zeroing only its attention drives accuracy to chance, so labels are transferred from context to query only once, in layer 1. Message passing matters only early and only on homophilous graphs, where zeroing it at layer 3 lowers Cora from 77.8% to 47.1% and the group mean from 68.8% to 57.8%. Every other single-branch removal, including any later attention block, changes final accuracy by at most 2%. Repeating or swapping layers leads to the same conclusion (Appendix A).

Takeaway 5. Only two computations are essential, the first layer's attention, which carries the labels to the queries, and early message passing on homophilous graphs.

⑥ REPAIR Self-repair. A small drop in final accuracy cannot distinguish an idle layer from one whose computation is redone by later layers. Only the latter are interchangeable, as looping requires. We therefore zero one branch of layer m and read the tuned decoders at every later layer (Fig. 7). The largest drop after layer m measures the damage, and the drop that remains at layer 12 measures what later layers fail to repair. Attention removed at layer 2 or 3 drops early-exit accuracy to 22–30% at the next layer or two, yet by layer 12 it is within 2% of the unchanged model, whereas attention removed at layer 1 is never repaired. Message passing removed in layers 2–4 on homophilous graphs is only partly repaired, and for layer 3 about half of the largest drop remains at layer 12.

Takeaway 6. Later layers redo a removed attention block's work, so one looped block can replace them. A removed hop is only partly repaired, so the number of hops depends on the graph.

4 Looped NodePFN WITH IN-CONTEXT DEPTH SELECTION

§3 suggests *looping in context*, in which one block replaces twelve layers and the labeled context determines the number of iterations (§4.2–4.3), and §4.1 shows why no fixed number is optimal for every graph.

4.1 WHY NO FIXED DEPTH IS OPTIMAL

Proposition 1 (informal; formal statement and proof sketch in Appendix D)**.** *Let G be a two-class cSBM (Deshpande et al., 2018) with homophily h and expected degree d, with Gaussian node features whose class means are at $\pm\mu$ noise standard deviations, and let SNR_k be the class separability of $\hat{A}^k X$ after k propagation steps. With $\lambda := 2h - 1$ and $\kappa := |\lambda|\sqrt{d}$,*

$$\mathrm{SNR}_k \;=\; \frac{2\mu\,\kappa^k}{\sqrt{1+\mu^2(1-\lambda^2)\sum_{j<k}\kappa^{2j}}}\,\big(1+o(1)\big) \tag{2}$$

below the mixing depth. With $\kappa_c := \sqrt{1+\mu^2(1-\lambda^2)}$, (i) if $\kappa < \kappa_c$ the best depth is $k = 0$, (ii) if $\kappa > \kappa_c$, SNR_k increases with k until the graph mixes, and (iii) for any prior spanning both regimes, no fixed k maximizes expected separability.

Each propagation step scales the class-mean component by $2h - 1$ and shrinks the feature noise by $1/\sqrt{d}$, as Baranwal et al. (2021) show for a single step, but it also adds graph noise, the sum in the denominator of Eq. (2). Whether a step is beneficial thus depends only on κ (§5, Fig. 8c, d).

4.2 ARCHITECTURE AND TRAINING

Looped NodePFN replaces the $L = 12$ distinct layers with a single block f_θ applied T times, $H^{(t+1)} = f_\theta(H^{(t)}, A)$. We call each application an *iteration* and the construction *looping*. Sharing the block gives 2.97M parameters against 29.0M, one tenth rather than one twelfth because the encoder and decoder are not looped. We attach the shared decoder to every iteration and minimize the average per-iteration PFN loss,

$$\mathcal{L}(\theta) = \mathbb{E}_{D\sim p(D)}\Big[\tfrac{1}{T}\sum_{t=1}^{T}\mathrm{CE}\big(\mathrm{dec}(H^{(t)}),\, y_{\text{test}}\big)\Big], \tag{3}$$

so that every iteration produces a calibrated prediction. Training uses NodePFN's prior, schedule and compute (30 epochs $\times$ 1024 steps, $T = 12$; about 9 GPU-hours on one A6000). This one-layer block looped twelve times is Loop-1L. We also train a two-layer block looped six times (Loop-2L, 5.33M) and, as a control, a one-layer model without looping (NodePFN-1L). Each model is trained once.

4.3 IN-CONTEXT DEPTH SELECTION (ICDS)

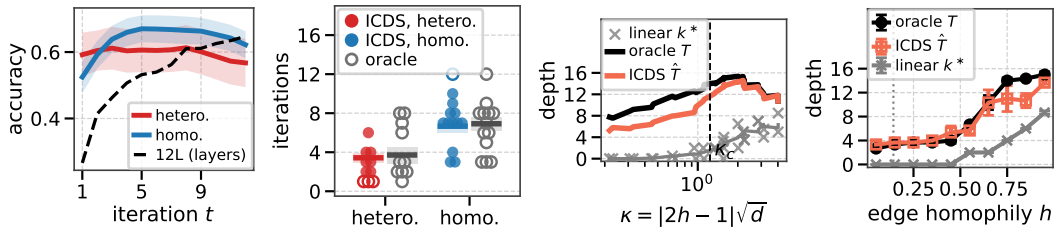
Because every iteration yields a prediction, T can be chosen without gradients. Given a graph with labeled context V_{train}, we hold out a fraction $q = 0.2$ of V_{train} as pseudo-queries, run the looped block for $T_{\max}$ iterations with the remaining context, and score each iteration by accuracy on the held-out nodes. $\hat{T}$ is the argmax of the curve averaged over 20 random splits (ties favor the smaller T), and the final prediction uses the full context at depth $\hat{T}$. The real query nodes *stay in the sequence* as unlabeled rows. They are valid in-context input (features and edges without labels), and removing them leads to too small a depth on sparsely labeled homophilous graphs (Appendix E). §5 uses the same rule to select, jointly with T, the number g of iterations that retain message passing. ICDS adds no parameters and costs $20\,T_{\max}$ block applications once per graph (2.7 s on the median graph), after which prediction is 2.2 times faster than with NodePFN-12L on the median graph (Appendix E). ICDS thus amounts to cross-validating one hyperparameter *within* the forward pass of a PFN.

5 EXPERIMENTS

Setup. Datasets, splits and the single-pass configuration follow §3, under both preprocessing conditions. Table 1 reports NodePFN-12L, the one-layer control NodePFN-1L, and Looped NodePFN in

Table 1: Accuracy (%) on the 23 benchmarks, mean ± std over three splits. Looped NodePFN with a one-layer (Loop-1L) and a two-layer block (Loop-2L) at their training depth (fixed T), with ICDS, and at the depth chosen with test labels (oracle T, gray, unranked). Schedule: message passing in the first g iterations only, (T, g) chosen by ICDS or by an oracle (§5). Best bold, second underlined.

	NodePFN		Loop-1L×12		Loop-2L×6		Loop-1L	Loop-1L + schedule	
Graphs	12L	1L	fixed T	ICDS	fixed T	ICDS	oracle T	ICDS	oracle
Params (M)	29.0	2.97	2.97	2.97	5.33	5.33	2.97	2.97	2.97
Standard (NodePFN k)									
All (23)	**66.6±0.7**	41.5±1.5	64.5±0.2	66.3±0.3	65.5±0.2	66.5±0.3	67.8±0.2	66.2±0.1	69.0±0.0
Homophilous (13)	68.5±1.0	36.8±1.0	67.4±0.4	69.3±0.3	69.2±0.1	**70.1±0.7**	70.6±0.5	68.5±0.8	71.6±0.5
Heterophilous (10)	**64.2±0.4**	47.7±2.1	60.7±0.2	62.3±0.7	60.8±0.4	61.8±0.4	64.1±0.3	63.3±0.9	65.7±0.6
No smoothing (k=0)									
All (23)	**65.0±0.6**	39.7±1.0	63.5±0.1	64.4±0.4	64.6±0.5	**65.0±0.5**	66.4±0.2	64.9±0.4	67.2±0.5
Homophilous (13)	66.0±0.7	33.2±0.4	65.6±0.0	66.2±0.1	**67.8±0.7**	67.6±0.6	68.1±0.3	66.4±0.2	68.4±0.3
Heterophilous (10)	**63.7±0.5**	48.1±2.3	60.6±0.2	62.1±0.9	60.5±0.5	61.6±0.5	64.0±0.3	63.1±1.0	65.5±0.8



(a) accuracy per iteration (b) chosen depth (c) cSBM: depth vs. κ (d) syn-Cora: depth vs. h

Figure 8: **The chosen depth follows the graph.** (a, b) Loop-2L on the benchmarks without smoothing, mean ± s.e., with NodePFN-12L's early exit dashed in (a). (c, d) Loop-1L on cSBM, with the dashed line at κ_c, and on syn-Cora, mean ± s.e. over three rewirings. Other panels in Appendix E.

its two forms, Loop-1L and Loop-2L, at their training depth (fixed T, 12 and 6 iterations), with ICDS, and with an *oracle* that selects per graph the depth maximizing test accuracy. The last two columns are discussed below, and Appendix E reports per-dataset results and other depth rules.

One block is nearly sufficient. Under the standard protocol ICDS brings Loop-1L to 66.3% and Loop-2L to 66.5%, compared with 66.6% for NodePFN-12L, and the oracle depth brings them to 67.8% and 68.3% (Table 1). The looped models are therefore limited by the choice of depth.

Looping is a training-time property. NodePFN-1L has the same 2.97M parameters as Loop-1L but was trained without looping. It reaches 41.5%. Surprisingly, looping it at inference drives its accuracy below 1% by t=6 (Fig. 13j). The trained looped models do not collapse. Their accuracy peaks near the training depth and declines slowly, from 65.8% at t=7 to 62.6% at t=16 for Loop-1L (Fig. 8a). This is the "overthinking" of recurrent networks (Bansal et al., 2022), explained here by the extra smoothing hop of each iteration (Fig. 14). Searching over T is therefore worthwhile.

ICDS improves accuracy. Under the standard protocol ICDS raises the mean accuracy of Loop-1L from 64.5% at the training depth to 66.3% and that of Loop-2L from 65.5% to 66.5% (Wilcoxon $p<10^{-5}$ and 0.015 over 69 graph–split pairs). Without smoothing the gains are similar and come from the heterophilous graphs, where the training depth over-propagates.

Does the selected depth track homophily? Figure 8b plots $\hat{T}$ against h for Loop-2L without pre-smoothing. The oracle depth increases with h, from 4.6 iterations on the heterophilous graphs to 6.4 on the homophilous ones, and ICDS follows it with more noise (5.1 vs. 6.5, Loop-1L in Appendix E). The scatter arises from the twenty labels per class, and the cSBM and syn-Cora results below, with more labels, are less noisy. Under the standard protocol the depth does not vary with h, because the pipeline has already propagated the homophilous graphs, as §3.2 found for NodePFN-12L.

On cSBM and syn-Cora the depth follows Proposition 1. A propagation step scales the class signal by $\kappa = |2h - 1|\sqrt{d}$ but also adds graph noise, so below κ_c the best depth is zero and above it the best depth increases with κ. On two-class cSBM graphs (Fig. 8c) the selected depth jumps at κ_c and then rises with κ, for ICDS and for the oracle alike. On syn-Cora, Cora with its edges rewired to a target homophily (Appendix D), the seven classes place every $h < 0.5$ below the threshold, and the selected depth rises with h (Fig. 8d). Choosing the depth in context raises accuracy from 60.2% to 63.2%, and from 47.7% to 53.3% on the heterophilous half, where twelve iterations oversmooth.

Why the looped block is less accurate on heterophilous graphs. The looped models are more accurate than `NodePFN-12L` on homophilous graphs and less accurate on heterophilous ones, where even the oracle depth does no better (Table 1). On those graphs the useful computation of `NodePFN-12L` is the label transfer of layer 1 and its later message-passing branches are nearly inactive (§3.3), which a shared block cannot express because every iteration propagates. We therefore let the block propagate only in its first g iterations and choose g together with T. With an oracle for (T, g), `Loop-1L` is more accurate than `NodePFN-12L` in every group, most on the heterophilous graphs (last column of Table 1). Here T controls the amount of refinement (*how long to think*) and g the receptive field (*how far to look*), and selecting g in context recovers only part of this gain.

6 Related work

Mechanistic studies of depth. Logit lenses (nostalgebraist, 2020; Belrose et al., 2023) and studies of layer skipping in language models (Sun et al., 2025; Lad et al., 2025) analyze depth layer by layer, and Balef et al. (2026) did so for tabular foundation models with a fixed loop of six. We follow their protocol and add a branch-wise ablation, the Dirichlet energy and a homophily probe. The middle-layer redundancy they report holds here only for refinement, because the graph determines how much propagation is needed. We are not aware of a comparable study of a graph foundation model.

Looped and recurrent transformers. Universal and looped transformers (Dehghani et al., 2019; Giannou et al., 2023; Saunshi et al., 2025) and recurrent reasoning models (Geiping et al., 2025; Wang et al., 2025; Jolicoeur-Martineau, 2025; Jeddi et al., 2026; Bae et al., 2025) iterate a small block, and some of them supervise every iteration, as we do, or allocate more iterations to harder inputs. In our setting an iteration is also a hop, so the useful number of iterations is bounded by the graph and must be selected per graph instead of being scaled (Proposition 1, Fig. 8a).

Choosing the depth. Adaptive-depth GNNs (Chien et al., 2021; Spinelli et al., 2021; Hevapathige et al., 2026) and halting rules (Graves, 2016; Banino et al., 2021; Popescu et al., 2026; Bae et al., 2025) learn the depth from training data. ICDS requires no learned selector, because a PFN has labeled context at test time and can use it to score its own iterations.

Scope. Our claims concern node classification with PFNs for graphs in the single-pass setting. The mechanistic results are established on `NodePFN` and reproduced on `GraphPFN`, which we do not loop because its backbone is frozen. Appendix F discusses these connections in more detail.

7 Conclusion

We asked how small a PFN for graphs can be and whether its computation can be adapted to the input graph. Depth provides refinement, which a single block can supply, and propagation, whose amount depends on the graph. `Looped NodePFN` therefore uses one block whose number of iterations is selected in context. With one-tenth of the parameters, it achieves 66.3% mean accuracy, compared with 66.6% for `NodePFN-12L`, and predicts up to 5.1 times faster once its depth is selected (2.2 times on the median graph). **Threats to validity.** The depth ICDS selects follows homophily only without pre-smoothing, and Proposition 1 covers the two-class cSBM (Appendix G). **Limitations and future work.** ICDS makes a single prediction slower than one of `NodePFN-12L`, less so with fewer hold-outs, and needs roughly fifty labeled nodes. The oracle results for per-node depth and for the branch schedule suggest further gains from adaptive computation, and by letting the labeled context select the number of iterations, this work opens the way for bringing the adaptive computation of looped transformers into PFNs.

AI USE STATEMENT

In this work, we used generative AI tools (Claude Code) to help implement our methods, namely the analysis code, the looped models, ICDS and the branch schedule, whose ideas come from the authors. The existing implementations we build on (`NodePFN`, `GraphPFN` and the cSBM and syn-Cora generators) were chosen and integrated by the authors. We also used these tools to assist in deriving and writing the proof of Proposition 1, whose statement and final proof were decided and written by the authors. The authors designed and ran all experiments, including the branch schedule. We used generative AI tools to help interpret results and to give feedback on the analysis, by summarizing results and identifying missing analyses, which the authors then addressed with their own experiments. We used DeepL, which includes an LLM-based feature, and Google Translate to assist with translation. We have not used generative AI tools to propose hypotheses, since the hypothesis that prediction depth grows with homophily was fixed by the authors before the experiments, or to generate synthetic data sets, since cSBM and syn-Cora come from existing generators. Cleaning or reformatting datasets and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to write and edit code, to create figures and tables, to draft and edit parts of the paper for readability, and to search the literature. We have reviewed all AI-assisted work. The implementation was verified by reproducing the official `NodePFN` predictions to 10^{-4} before any experiment (Appendix A), every number in the paper is produced by our code from saved runs, and the proof was checked by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only public benchmark graphs and synthetic graphs and involves no human subjects. The social and crowdsourcing networks among the benchmarks were released in anonymized form by their creators, and we use them as distributed. We do not foresee strongly negative societal impacts specific to this work. As a general method for node classification, it carries the same risks as the applications in which it is used. A looped block requires one-tenth of the parameters of the model it replaces, which reduces the memory required to store and train a PFN for graphs, although ICDS adds computation at inference time (Appendix G).

REPRODUCIBILITY STATEMENT

We release anonymized code that reproduces Table 1 (`https://anonymous.4open.science/r/looped-nodepfn`). It contains the looped model, ICDS and the code to pretrain the looped models, the checkpoints of `Loop-1L`, `Loop-2L` and `NodePFN-1L` (`NodePFN-12L` is the public checkpoint of Choi et al. (2026)), and the per-graph results behind every cell of the table, and it evaluates every column under both preprocessing conditions on all three splits. The code of the layer-wise analysis in §3 will be released with the camera-ready version. Our analysis runs on the public `NodePFN` code and reproduces its official predictions to 10^{-4} before any experiment (Appendix A). The architecture, training objective and training settings are in §4.2 and Appendix E, and ICDS is specified in §4.3. The benchmarks, splits and both preprocessing conditions are described in §3 and Appendices A and B, and the synthetic graphs in Appendix D. Appendix D also gives the full statement and proof of Proposition 1 with its assumptions and a numerical check, and Appendix E reports the computational cost. Each model was trained once and evaluated on three data splits (Appendix G).

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, 2025.

Amir Rezaei Balef, Mykhailo Koshil, and Katharina Eggensperger. Is one layer enough? understanding inference dynamics in tabular foundation models. In *ICML*, 2026.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. In *ICML Workshop*, 2021.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, 2022.

Aseem Baranwal, Kimon Fountoulakis, and Aukosh Jagannath. Graph convolution for semi-supervised classification: Improved linear separability and out-of-distribution generalization. In *ICML*, 2021.

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 2021.

Jeongwhan Choi, Jongwoo Kim, Woosung Kang, and Noseong Park. Learning posterior predictive distributions for node classification from synthetic graph priors. In *ICLR*, 2026.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *ICLR*, 2019.

Yash Deshpande, Subhabrata Sen, Andrea Montanari, and Elchanan Mossel. Contextual stochastic block models. In *Advances in Neural Information Processing Systems*, 2018.

Carson Dudley and Samet Oymak. Latent chain-of-thought improves structured-data transformers. *arXiv preprint arXiv:2605.11262*, 2026.

Dmitry Eremeev, Oleg Platonov, Gleb Bazhenov, Artem Babenko, and Liudmila Prokhorenkova. Graphpfn: A prior-data fitted graph foundation model. In *ICML*, 2026.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *ICML*, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Asela Hevapathige, Asiri Wijesinghe, and Ahad N. Zehmakan. Beyond fixed depth: Adaptive graph neural networks for node classification under varying homophily. *AAAI Conference on Artificial Intelligence*, 2026. arXiv:2511.06608.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. In *ICLR*, 2023.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637:319–326, 2025.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2602.11451.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. The remarkable robustness of llms: Stages of inference? *arXiv preprint arXiv:2406.19384*, 2025.

Jing Lei and Alessandro Rinaldo. Consistency of spectral clustering in stochastic block models. *The Annals of Statistics*, 43(1):215–237, 2015.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One for all: Towards training one graph model for all classification tasks. In *ICLR*, 2024.

Sitao Luan, Chenqing Hua, Minkai Xu, Qincheng Lu, Jiaqi Zhu, Xiao-Wen Chang, Jie Fu, Jure Leskovec, and Doina Precup. When do graph neural networks help with node classification? investigating the homophily principle on node distinguishability. In *NeurIPS*, 2023.

Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *ICLR*, 2022.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *ICLR*, 2022.

nostalgebraist. Interpreting gpt: the logit lens. LessWrong, 2020.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020.

Oleg Platonov, Gleb Bazhenov, Dmitry Eremeev, and Liudmila Prokhorenkova. A fair evaluation of graph foundation models for node property prediction. *ICML 2026 Workshop on Graph Foundation Models*, 2026. arXiv:2606.24509.

Andrei Cristian Popescu, Haitz Sáez de Ocáriz Borde, and Pietro Liò. Adaptive depth in looped transformers: Diagnosing learned halting gates and trajectory readouts. *arXiv preprint arXiv:2607.20519*, 2026.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. Tabicl: A tabular foundation model for in-context learning on large data. In *ICML*, 2025.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, 2022.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025.

Indro Spinelli, Simone Scardapane, and Aurelio Uncini. Adaptive propagation graph convolutional network. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10):4755–4760, 2021.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. Transformer layers as painters. *AAAI*, 2025.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *SIGIR*, 2024.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Jianan Zhao, Zhaocheng Zhu, Mikhail Galkin, Hesham Mostafa, Michael Bronstein, and Jian Tang. Fully-inductive node classification on arbitrary graphs. In *ICLR*, 2025.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Advances in Neural Information Processing Systems*, 2020.

Appendix

A Details of the mechanistic study and per-dataset results 13

B Feature pre-smoothing and prediction depth 15

C GraphPFN 16

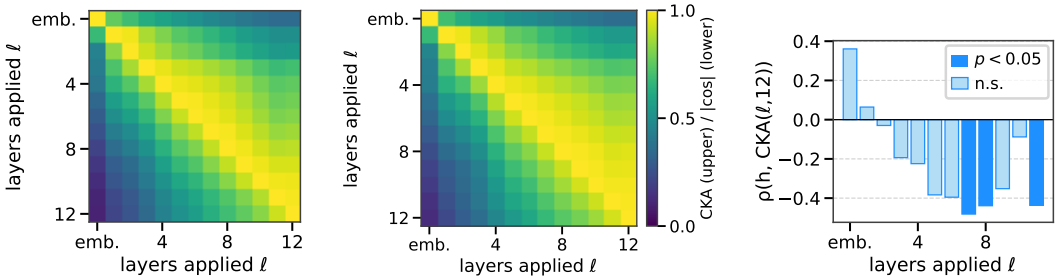
D Proof of Proposition 1 17

E Details of `Looped NodePFN` 19

F Related work in detail 23

G Threats to validity and limitations 25

A Details of the mechanistic study and per-dataset results



(a) homophilous (b) heterophilous (c) correlation with h per layer

Figure 9: ① SIM **Full similarity matrices.** Layer-wise linear CKA (upper triangle) and mean absolute cosine (lower triangle) between query representations, averaged over (a) homophilous and (b) heterophilous graphs. (c) Spearman correlation between h and $\mathrm{CKA}(\ell, 12)$ per layer (filled: $p < 0.05$).

PFN objective. A PFN f_θ is trained on datasets $D \sim p(D)$ to minimize $\mathbb{E}_D[-\log q_\theta(y_{\text{test}} \mid x_{\text{test}}, D_{\text{train}})]$, and thus approximates the posterior predictive distribution (PPD) of the prior (Müller et al., 2022). At test time a real dataset is passed as context and predictions are produced in one forward pass without gradient updates.

Configuration. Every analysis uses a single forward pass with one ensemble member, no class or feature permutation and the identity preprocessing member. Feature smoothing, truncated SVD, normalization and splits come from the official inference code, whose inputs we intercept rather than re-implement, and our re-implementation of the layers reproduces the official predictions to 10^{-4}. Single-pass accuracy is below the ensembled numbers reported by Choi et al. (2026), and every model in this paper is evaluated in this configuration.

Conventions. Accuracy is in %, and a difference between two accuracies is in the same unit. Layers are numbered $1, \ldots, 12$, and $H^{(0)}$ is the input embedding ("emb." on the axes where it is a meaningful control). ρ is over the 19 informative graphs unless stated, and correlations over all 23 graphs or pooled over graph–split pairs are marked as such. The four majority-level graphs are reported in ROC-AUC in Table 2.

Repeat and swap. Repeating layer 1 lowers final accuracy from 68.8% to 31.5% on homophilous graphs and from 63.6% to 50.7% on heterophilous ones, and swapping it with layer 2 lowers them to 37.0% and 47.8%. Repeating layer 4 or swapping layers 2–3 lowers homophilous accuracy to

54.5% and 53.8%. From layer 5 onward, repeating or swapping layers has little effect in either group (Fig. 6).

Tuned decoders. Following Balef et al. (2026), we train 13 heads, one per state $H^{(0)}, \ldots, H^{(12)}$, initialized from the model's decoder with the backbone frozen. They are trained for 2k steps on 256 graphs of 1024 nodes drawn from NodePFN's prior with the training-time sampler (2 to 20 classes, up to 100 features). Trained instead on the context halves of the evaluation graphs, the $H^{(0)}$ head on Cora reaches 72% and ℓ^* collapses to 0 on 4 of 19 graphs, so the head learns the target task rather than reading the representation.

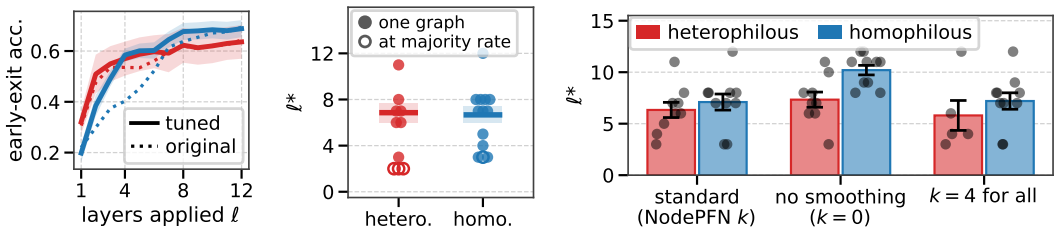
Majority-level graphs. deezer, minesweeper, tolokers and questions have single-pass accuracy within 1% of the majority class. Their single-pass ROC-AUC is 0.50, 0.75, 0.71 and 0.63. They are excluded from every accuracy-dependent correlation and drawn hollow.

Threshold sensitivity of ℓ_{rep}. Over all 23 graphs (ℓ_{rep} does not depend on accuracy), ℓ_{rep} rises with h at every CKA threshold and significantly at 0.9 (ρ=0.33, 0.48 and 0.30 with p=0.13, 0.02 and 0.17 at CKA thresholds 0.8, 0.9 and 0.95), and the threshold-free mean of $\mathrm{CKA}(\ell, 12)$ over $\ell < 12$ does not vary with h ($\rho = -0.22$, p=0.31).

Per-dataset results. Table 2 lists every graph on split 0 under the standard protocol, with NodePFN-12L, each looped model and the one-layer control at its training depth, with ICDS and with the oracle depth. The four majority-level graphs are reported in ROC-AUC (†).

Table 2: Per-dataset single-pass accuracy (%; †ROC-AUC) on split 0, standard protocol. Columns are NodePFN-12L, then Loop-1L, Loop-2L and NodePFN-1L at the training depth (fixed), with ICDS and with the oracle depth, then Loop-1L with the branch schedule of §5, (T, g) chosen by ICDS or by an oracle. Bold rows are group and overall means.

	NodePFN-12L	Loop-1L			Loop-2L			NodePFN-1L			Loop-1L + schedule	
Dataset	12L	fixed	ICDS	oracle	fixed	ICDS	oracle	fixed	ICDS	oracle	ICDS	oracle
Homophilous												
air-brazil	76.92	65.38	69.23	69.23	61.54	73.08	73.08	26.92	26.92	26.92	69.23	69.23
air-europe	51.25	47.50	47.50	48.75	45.62	46.88	48.12	32.50	32.50	32.50	47.50	52.50
air-usa	58.02	56.22	53.69	59.28	58.74	59.10	59.46	37.84	37.84	37.84	52.07	59.28
amazon-computer	72.96	68.70	73.95	74.93	71.89	74.38	74.38	28.17	28.17	28.17	73.95	74.93
amazon-photo	83.82	83.87	83.15	85.39	85.37	85.53	85.66	38.93	38.93	38.93	83.15	85.85
citeseer	60.70	62.50	65.20	65.20	67.70	68.10	68.40	25.40	25.40	25.40	65.30	65.30
coauthor-cs	55.20	74.49	77.75	77.75	72.81	82.82	82.82	26.16	26.16	26.16	77.75	77.75
coauthor-physics	92.74	91.46	92.00	93.18	92.78	92.95	92.95	40.75	40.75	40.75	92.00	93.18
cora	79.60	77.80	79.10	79.80	75.30	77.00	79.50	33.70	33.70	33.70	74.90	79.80
dblp	71.72	64.04	71.78	71.78	67.94	68.86	72.82	45.21	45.21	51.01	67.75	71.78
deezer†	50.29	49.10	48.60	49.71	51.15	50.77	51.15	51.42	51.42	51.42	48.60	51.04
pubmed	77.60	73.40	74.00	74.10	75.20	73.50	75.60	50.90	50.90	50.90	72.30	76.90
wikics	75.32	68.87	70.41	70.75	75.13	75.13	75.13	27.72	27.72	27.72	70.41	70.75
Avg. homoph.	69.70	67.95	69.72	70.76	69.32	71.39	72.24	35.82	35.82	36.26	68.84	71.41
Heterophilous												
actor	31.84	29.54	31.32	32.70	27.17	33.22	33.22	24.08	24.08	24.08	34.21	34.41
amazon-ratings	43.95	41.25	43.67	44.21	43.02	44.44	44.44	41.42	41.42	41.42	43.67	44.21
chameleon	45.88	47.94	46.91	50.00	40.72	39.69	42.78	29.90	25.77	29.90	46.91	50.00
cornell	70.27	54.05	62.16	64.86	67.57	59.46	67.57	29.73	29.73	29.73	54.05	70.27
minesweeper†	74.87	79.24	79.24	80.52	79.12	78.44	79.12	68.37	68.37	68.37	82.64	82.64
questions†	62.74	43.46	43.03	49.53	44.86	45.66	58.67	49.44	49.44	58.19	43.03	57.06
squirrel	43.36	42.48	42.70	42.70	39.60	40.27	41.59	28.32	35.84	35.84	42.70	42.70
texas	78.38	62.16	64.86	72.97	67.57	72.97	75.68	13.51	13.51	13.51	75.68	75.68
tolokers†	70.96	68.46	74.03	74.03	71.68	72.65	73.41	67.55	67.55	67.55	74.03	74.03
wisconsin	78.43	70.59	70.59	72.55	66.67	66.67	72.55	37.25	37.25	37.25	70.59	80.39
Avg. hetero.	60.07	53.92	55.85	58.41	54.80	55.35	58.90	38.96	39.30	40.58	56.75	61.14
Avg. all	65.51	61.85	63.69	65.39	63.01	64.42	66.44	37.18	37.33	38.14	63.58	66.94



(a) NodePFN: accuracy, standard (b) NodePFN: ℓ^*, standard (c) ℓ^* by preprocessing

Figure 10: **Pre-smoothing performs the propagation.** (a, b) The logit lens of Fig. 5a, b under the standard protocol, where the two groups coincide and ℓ^* does not vary with h (ρ=0.13, p=0.59). (c) ℓ^* by group (bars: mean $\pm$ s.e., dots: graphs) under the standard protocol and with k=0 or k=4 for every graph. ℓ^* follows h only with k=0 (ρ=0.13, 0.57 and 0.23, with n=15 under k=4, where four heterophilous graphs fall to the majority class).

B Feature pre-smoothing and prediction depth

The standard protocol. NodePFN's official pipeline smooths the features k times before the model, with k set per dataset. It is four for Cora and Coauthor-Physics, three for the Amazon and DBLP graphs and zero for the WebKB, Actor and Chameleon graphs, so k itself rises with h (ρ=0.75 over all 23 graphs, $p<10^{-4}$). The public GraphPFN evaluation (Eremeev et al., 2026; Platonov et al., 2026) tunes only optimizer settings per dataset and has a fixed message-passing depth.

Removing the pre-smoothing. We rerun the logit lens with the same k for every graph (Figs. 5 and 10). With k=0 the model performs all of the propagation, and the prediction holds. ℓ^* rises with h on each of the three splits (ρ=0.57, 0.62 and 0.51, $p \leq 0.03$ each, and 0.56 with $p<10^{-4}$ pooled over the 57 informative graph–split pairs), from 7.0 layers on heterophilous graphs to 10.0 on homophilous ones. Under the standard protocol it does not vary with h on any split (ρ=0.13, 0.00 and 0.17). Only the homophilous graphs change, by 2.9 layers for 2.6 removed smoothing steps on average, since the heterophilous graphs already had k=0.

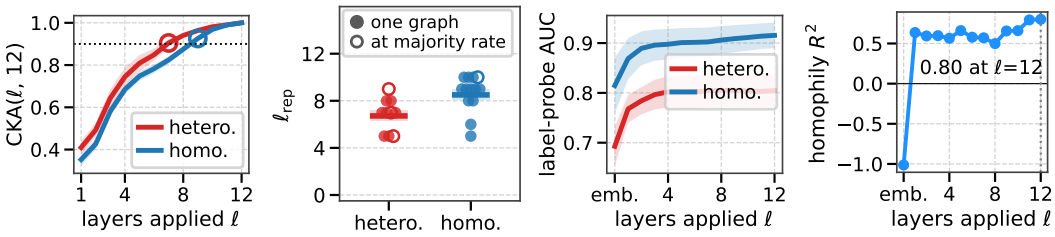
Too much smoothing. With k=4 for every graph ℓ^* again does not follow h, and four heterophilous graphs drop to majority-class accuracy, so the model cannot undo smoothing applied before it.

The official pipeline. The official pipeline (32-member ensemble with class and feature permutations, five splits) shows the same effect. Setting k=0 lowers Cora from 82.1% to 76.2% and DBLP from 77.4% to 68.8%. PubMed, whose protocol k is two, changes only within noise (78.6% to 79.7%), and Texas, whose k is already zero, is unchanged (74.6%). The pre-smoothing thus supplies propagation that the fixed-depth model does not, which accounts for several percent of accuracy on homophilous graphs.

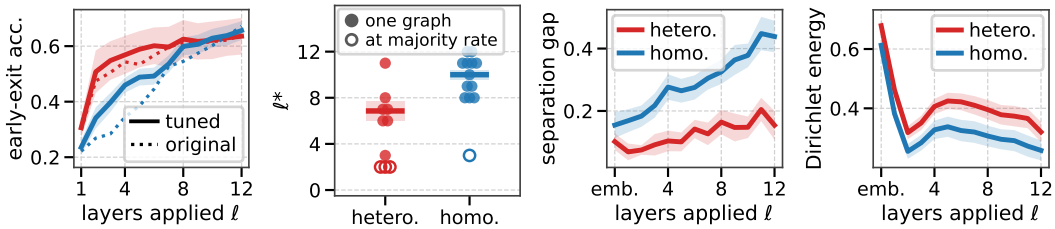
The same dependence in GraphPFN. GraphPFN's pipeline does no pre-smoothing (Appendix C), and the dependence appears under its own protocol. Its lens, a linear probe of the query labels on the target-token state of each layer, reaches 95% of its final value at 5.1 layers on heterophilous graphs and 9.6 on homophilous ones (ρ=0.52, p=0.032, n=17, Fig. 5c, d), values that NodePFN reaches only without pre-smoothing.

The readouts of §3 without pre-smoothing. Figure 11 repeats the NodePFN panels of Figs. 3 and 4 with k=0 for every graph (split 0), together with the lens panels of Fig. 5a, b. Only the quantities that §3.2 predicts to change are affected. Labels are still transferred in layer 1, the homophily probe explains 64% of the variance of h after layer 1 (80% at its peak), and the Dirichlet dip on heterophilous graphs is unchanged. The prediction depth now follows h (panels e, f). The representation-convergence depth ℓ_{rep} follows it in the same direction, 6.8 layers on heterophilous against 8.6 on homophilous graphs (panel b), but its dependence on h changes much less than that

of ℓ^* (ρ=0.48 under the standard protocol, 0.48 after partialling out k, 0.60 with k=0, over all 23 graphs), consistent with it measuring a different quantity.



(a) CKA to the final layer (b) ℓ_{rep} by group (c) label probe (d) homophily probe



(e) early-exit accuracy (f) ℓ^* by group (g) separation gap (h) Dirichlet energy

Figure 11: NodePFN without pre-smoothing (k=0 for every graph, split 0), in the designs of the main-text figures. ℓ_{rep} (b) and ℓ^* (f) rise with h (ρ=0.60, p=0.003 over all 23 graphs, and ρ=0.57, p=0.011 over the informative graphs).

C GraphPFN

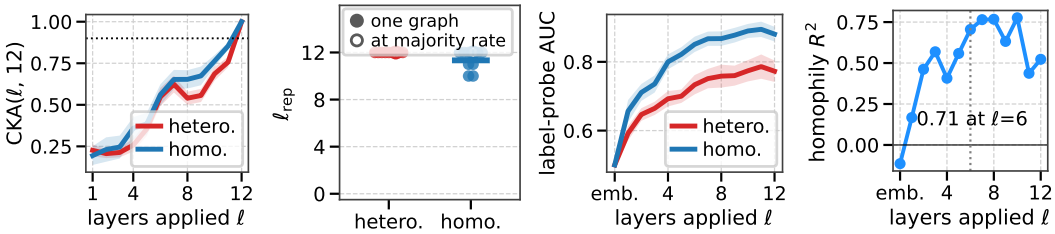
Setup. GraphPFN-1.3 (Eremeev et al., 2026) wraps each of the 12 layers of a frozen LimiX-16M with a graph-attention module and an MLP, both residual and zero-initialized at the start of its pretraining, and only these adapters were trained. We run the released checkpoint through its own inference pipeline (standardized features, no PCA, no smoothing, the graph made simple and bidirected) with a single ensemble member and no feature or class permutation, on the same split 0 as §3. Three graphs are excluded. The graph coauthor-cs has 15 classes and GraphPFN requires at least two ensemble members above ten classes, and coauthor-physics and questions exceed GPU memory because LimiX tokenizes every feature of every node. Texas and deezer are at or below the majority rate for a single member (the ten-member ensemble recovers Texas, 70%) and are shown hollow and excluded from correlations, leaving $n = 17$.

Readout. LimiX reads its prediction from a target-token column. For query rows that token starts as a constant "missing target" embedding and only acquires the prediction through attention over rows and features. GraphPFN's own decoder therefore yields no usable prediction before layer 10 (Fig. 12c), and a label-free affine translator to the final layer (the tuned lens of Belrose et al., 2023, fitted leave-one-graph-out) is unstable on these states. We instead use a two-fold linear probe of the query labels on the target-token state of each layer, fitted and scored on disjoint halves of the query nodes of the same graph, and define ℓ^* on its curve. On NodePFN the analogous probe is at ceiling from layer 1 (Fig. 3c), which is itself a difference between the two architectures. NodePFN's attention transfers labels to the query representation in one step, whereas LimiX accumulates them.

Interventions. Fig. 12 gives final accuracy when a whole wrapped layer is skipped or only its graph adapter is zeroed (the residual stream is kept), averaged over the informative graphs of each group. As in NodePFN, skipping layer 1 lowers accuracy from 67.3% to 23.7% on homophilous graphs and from 62.4% to 28.5% on heterophilous ones. LimiX has a second essential region at layers 9–10, where skipping a layer lowers homophilous accuracy to 40.1% and 29.6% and heterophilous accuracy to 39.4% and 49.0%, and skipping any middle layer lowers accuracy (by 3 to 19%). Zeroing a single graph adapter lowers accuracy by at most 5.3%, and the losses sum to 32% on homophilous and 12% on heterophilous graphs, mostly in layers 4–10. Propagation is thus, as in

NodePFN, a modest and homophily-weighted addition to a label-transfer backbone, but it is spread over layers 4–10 instead of concentrated in two early layers, consistent with adapters trained on a frozen backbone.

Dirichlet energy. The Dirichlet energy of the target-token states stays flat across layers in both groups (Fig. 4d), with none of the early dip that NodePFN's message passing produces on heterophilous graphs, consistent with a graph branch whose effect is spread across layers.



(a) CKA to the final layer (b) ℓ_{rep} by group (c) label probe (d) homophily probe

Figure 12: GraphPFN, same designs as Fig. 3. (a) CKA between each layer's target-token representation and the final one, group mean ± s.e. (b) ℓ_{rep} by group, which does not vary with h (ρ=−0.13, p=0.62, n=17), so the representations keep changing until the last layer. (c) Query-label probe per layer, which rises over all twelve layers because LimiX accumulates label information instead of transferring it once. (d) homophily probe, which explains 70% of the variance of h by layer 3. Lens and ablation panels are in Figs. 5 and 6.

D Proof of Proposition 1

Setting. Nodes $i \in [n]$ carry balanced labels $y_i \in \{\pm 1\}$. Edges are independent, $A_{ij} \sim \text{Bern}(p)$ if $y_i = y_j$ and $\text{Bern}(q)$ otherwise, with p, q depending on n so that the expected degree $d = \frac{n}{2}(p+q)$ satisfies $d = \omega(\log n)$ and $d = n^{o(1)}$. The homophily is $h = p/(p+q)$, and we write $\lambda := 2h - 1 = (p-q)/(p+q) \in [-1, 1]$ and $\kappa := |\lambda|\sqrt{d}$. Features are $x_i = \mu\, y_i u + z_i$ with u a fixed unit vector and $z_i \sim \mathcal{N}(0, I_F)$ independent of the graph, so that along u the raw class means are $\pm\mu$ and the raw within-class standard deviation is 1. We propagate with the random-walk normalization $\hat{A} = D^{-1}A$. The symmetric normalization with self-loops used by the pipeline (§3) has the same two leading eigenvectors and replaces λ by $\lambda_d := (d\lambda+1)/(d+1) = \lambda + O(1/d)$, which is absorbed in the $o(1)$ below (but see Remark (b)). This is the contextual SBM of Deshpande et al. (2018) in the regime studied by Baranwal et al. (2021).

For a depth k let $H_k = \hat{A}^k X$ and $\pi_i^{(k)} := \langle (H_k)_i, u \rangle$ be the projection of node i's representation on the class-mean direction. Define

$$\text{SNR}_k := \frac{\left|\mathbb{E}[\pi_i^{(k)} \mid y_i = +1] - \mathbb{E}[\pi_i^{(k)} \mid y_i = -1]\right|}{\sqrt{\text{Var}[\pi_i^{(k)} \mid y_i]}}, \qquad \text{so } \text{SNR}_0 = 2\mu,$$

where expectation and variance are over the graph and the features. Since u is a fixed linear readout, SNR_k lower-bounds the Fisher discriminant ratio of the best linear readout of H_k. It is the quantity computed numerically in Fig. 15 (there u is estimated from the realized class means).

Proposition 1 (formal). Fix $k \geq 1$ with $d^k = o(n)$ and let $S_k := \sum_{j=0}^{k-1} \kappa^{2j}$, $\nu^2 := \mu^2(1-\lambda^2)$. Then, as $n \to \infty$,

$$\text{SNR}_k = \frac{2\mu\,\kappa^k}{\sqrt{1 + \nu^2 S_k}}\,\big(1 + o(1)\big).$$

Consequently, with $\kappa_c = \sqrt{1+\nu^2}$, (i) if $\kappa < \kappa_c$ then $\text{SNR}_k < \text{SNR}_0$ for every $k \geq 1$, (ii) if $\kappa > \kappa_c$ then $\text{SNR}_{k+1} > \text{SNR}_k$ for every $k \geq 0$ (within the range $d^{k+1} = o(n)$), with $\text{SNR}_k \to 2\sqrt{(\kappa^2-1)/(1-\lambda^2)}$, and (iii) no single k maximizes $\mathbb{E}_{(h,d)\sim\mathcal{P}}[\text{SNR}_k]$ for a prior $\mathcal{P}$ whose support meets both $\{\kappa < \kappa_c\}$ and $\{\kappa > \kappa_c\}$ with positive mass.

Proof sketch. The argument has three steps. First, the class-mean signal after k steps is λ^k times the original plus graph noise that shrinks by $1/\sqrt{d}$ per step (which gives the numerator and the $\nu^2 S_k$ term). Second, the feature noise shrinks by $1/\sqrt{d}$ per step as long as the k-hop neighborhoods are tree-like (the leading 1 in the denominator). Third, the ratio of consecutive SNR_k then depends on κ alone, which gives the threshold. Decompose $H_k = \mu\,(\hat{A}^k y)\,u^\top + \hat{A}^k Z$ with Z the noise matrix; along u,

$$\pi_i^{(k)} = \mu\,(\hat{A}^k y)_i + (\hat{A}^k \zeta)_i, \qquad \zeta := Zu \sim \mathcal{N}(0, I_n),$$

and the two terms are independent. We treat the signal $(\hat{A}^k y)_i$ and the noise $(\hat{A}^k \zeta)_i$ separately.

One step of signal. For node i let D_i^+ and D_i^- be its numbers of same- and other-class neighbors, $D_i^+ \sim \mathrm{Bin}(\frac{n}{2}, p)$, $D_i^- \sim \mathrm{Bin}(\frac{n}{2}, q)$, independent. Then $(\hat{A}y)_i = y_i\,(D_i^+ - D_i^-)/(D_i^+ + D_i^-)$. Its mean is $\lambda y_i\,(1 + O(1/d))$, and by the delta method (the degrees concentrate because $d = \omega(\log n)$) its variance is

$$\Big(\frac{2(1-h)}{d}\Big)^2 dh + \Big(\frac{2h}{d}\Big)^2 d(1-h) = \frac{4h(1-h)}{d} = \frac{1-\lambda^2}{d}.$$

The degree normalization cancels the common fluctuation of $D_i^+ + D_i^-$, which is why the one-step graph noise is $(1-\lambda^2)/d$ rather than $1/d$ and vanishes as $|\lambda| \to 1$; Baranwal et al. (2021) obtain the same one-step quantities.

k steps of signal. Write $\hat{A} = M + N$ with $M := \mathbb{E}[\hat{A}]$ and $N := \hat{A} - M$. Up to $O(1/d)$ corrections M has rank two, $M\mathbf{1} = \mathbf{1}$ and $My = \lambda y$, and $\|N\|_{\mathrm{op}} = O(1/\sqrt{d})$ with high probability (Lei & Rinaldo, 2015). Expand $\hat{A}^k y = (M+N)^k y$ into 2^k words. Ny is centered with per-coordinate standard deviation $\Theta(\sqrt{(1-\lambda^2)/d})$ (the one-step computation) and is nearly orthogonal to $\mathbf{1}$ and y, so applying M to any word that already contains an N produces a vector of per-coordinate size $O(1/\sqrt{nd})$, which is negligible. The surviving words are $N^j M^{k-j} y = \lambda^{k-j} N^j y$, $j = 0, \dots, k$. For $d^k = o(n)$ the k-step neighborhoods are tree-like, so $(N^j y)_i$ has mean $O(1/d)$, standard deviation $\Theta\big(\sqrt{1-\lambda^2}\,d^{-j/2}\big)$, and the different j are uncorrelated to leading order. Hence, conditionally on y_i,

$$(\hat{A}^k y)_i = \lambda^k y_i + \sum_{j=1}^{k} \lambda^{k-j}\,\xi_i^{(j)}, \qquad \mathbb{E}\,\xi_i^{(j)} = O(1/d), \quad \mathrm{Var}\,\xi_i^{(j)} = \frac{1-\lambda^2}{d^j}\,(1+o(1)),$$

so the class-mean separation of $\pi^{(k)}$ is $2\mu|\lambda|^k(1 + O(k/d))$ and the graph-induced within-class variance is $\mu^2(1-\lambda^2)\sum_{j=1}^k \lambda^{2(k-j)} d^{-j}(1+o(1)) = \nu^2 d^{-k} S_k\,(1+o(1))$, using $\lambda^2 = \kappa^2/d$.

Noise. Conditionally on the graph, $(\hat{A}^k\zeta)_i \sim \mathcal{N}\big(0, \|(\hat{A}^k)_{i\cdot}\|_2^2\big)$. For $d^k = o(n)$ the k-step neighborhood of i is tree-like with $\Theta(d^k)$ leaves each of weight $\Theta(d^{-k})$, so $\|(\hat{A}^k)_{i\cdot}\|_2^2 = d^{-k}(1+o(1))$. Adding the two independent variances and dividing gives the displayed formula since $|\lambda|^k = \kappa^k d^{-k/2}$.

Consequences. $\mathrm{SNR}_k^2 = 4\mu^2\kappa^{2k}/(1+\nu^2 S_k)$, so $\mathrm{SNR}_{k+1}^2/\mathrm{SNR}_k^2 = \kappa^2(1+\nu^2 S_k)/(1+\nu^2 S_k + \nu^2\kappa^{2k})$, which exceeds 1 iff $(\kappa^2-1)(1+\nu^2 S_k) > \nu^2\kappa^{2k}$, i.e. (using $(\kappa^2-1)S_k = \kappa^{2k} - 1$) iff $\kappa^2 - 1 > \nu^2$. The condition does not depend on k. For $\kappa > \kappa_c$ the sequence is strictly increasing, with limit $4\mu^2\kappa^{2k}/(\nu^2\kappa^{2k}/(\kappa^2-1)) = 4(\kappa^2-1)/(1-\lambda^2)$. For $\kappa < \kappa_c$ it is strictly decreasing from $\mathrm{SNR}_0 = 2\mu$. Claim (iii) follows because the maximizer of SNR_k is 0 on one event of positive prior mass and ≥ 1 on another, so no single k maximizes the expectation. □

Numerical check. On the cSBM graphs of Fig. 15 (n=1024, four seeds per cell, $\mu \approx 0.72$, and the pipeline's self-loops make λ_d the relevant contraction) the threshold rule "$\arg\max_k \mathrm{SNR}_k = 0$ iff $\kappa < \kappa_c$" holds in 26 of the 30 (h, d) cells, and every exception has κ within 0.2 of κ_c. The one-step formula is within 10% of the measured SNR_1 in 19 of the 21 cells with $\kappa > 1$ (median error 3%). The k-step formula is accurate while the k-hop neighborhoods are tree-like ($d \geq 10$, $\kappa < 3$, median error 10% at k=2), and the measured curves keep the threshold and the monotonicity of Eq. (2).

syn-Cora. To test the proposition beyond Gaussian features and two balanced classes, we use a semi-synthetic graph with real features. syn-Cora keeps Cora's node features (after the official

truncated SVD to 15 dimensions, with no smoothing), its seven classes and its degree sequence, and rewires the edges by degree-preserving swaps to a target homophily $h \in [0.05, 0.95]$, after Zhu et al. (2020). We draw three rewirings and 20-per-class splits for each h. With C classes the contraction per step is $\lambda_C = h - (1-h)/(C-1)$, so for C=7 every $h < 0.5$ lies below the threshold.

The looped model on cSBM, per cell. The cSBM grid covers $h \in [0.05, 0.95]$ and $d \in \{5, 10, 20\}$. The depth the linear theory prefers changes at κ_c, from 0.4 steps below to 4.2 above, and the depth ICDS selects rises from 7.4 to 13.1 iterations across κ_c. The ICDS depth rises with κ (ρ=0.62, $p < 10^{-3}$), and so does the oracle depth, more noisily where the accuracy curve is flat (ρ=0.37, p=0.045). The model exploits strong heterophily as the bound allows (99.8% at h=0.05, d=20).

The looped model on syn-Cora. The oracle depth follows h almost perfectly (ρ=1.00) and the ICDS depth nearly so (ρ=0.98). ICDS raises accuracy from 60.2% to 63.2% (Wilcoxon $p < 10^{-4}$), from 47.7% to 53.3% on the heterophilous half and only from 72.8% to 73.1% on the homophilous half, where twelve iterations are already about right.

Remarks.

(a) Beyond k_{mix} the rows of $\hat{A}^k$ equilibrate, the noise variance saturates at $\Theta(1/n)$ while the signal keeps contracting as $|\lambda|^k$, and SNR_k falls. This is oversmoothing, and it caps the useful depth even when $\kappa > \kappa_c$ (in Fig. 15a the d=20 graphs with the largest κ select the fewest iterations).

(b) For weak features ($\mu \to 0$, the regime of Baranwal et al., 2021) the threshold is $\kappa_c \to 1$. For Fig. 15 it is $\kappa_c = \sqrt{1 + \mu^2(1-\lambda^2)} \in [1.0, 1.23]$, and the dashed line is drawn at 1.23. With self-loops the effective $\lambda_d = (d\lambda + 1)/(d+1)$ is pulled towards $+1$, so heterophilous graphs cross the threshold slightly later than homophilous ones of the same κ (the triangles in Fig. 15a).

(c) For $h < \frac{1}{2}$ the signal $\lambda^k y$ alternates in sign with k. The separability bound is unaffected, but a readout shared across graphs and depths, as in a PFN decoder, cannot exploit odd steps on heterophilous graphs, one reason the learned model gains less from propagation at $h < \frac{1}{2}$ than the bound allows (Fig. 15a).

(d) With C balanced classes the expected adjacency has, besides $\mathbf{1}$, a $(C-1)$-dimensional class-contrast eigenspace with eigenvalue $\lambda_C = h - (1-h)/(C-1)$ (the same-class rate minus the rate of any one other class), and the argument goes through with λ replaced by λ_C and the one-step graph-noise variance by $\big(h(1-h) + \dots\big)/d = O(1/d)$, and the threshold becomes $|\lambda_C|\sqrt{d} > \kappa_c$. Since $|\lambda_C| \le 1/(C-1)$ for $h \le \frac{1}{2}$, heterophily is nearly unexploitable by propagation once $C \ge 3$. On syn-Cora (C=7, $d \approx 4$) every $h < 0.5$ has $|\lambda_C|\sqrt{d} < 0.4$ and $k^* = 0$, while $h = 0.95$ has $\lambda_C = 0.94$ and $k^* = 9$ (Fig. 15b).

E Details of Looped NodePFN

Training. We use NodePFN's prior and schedule, with deep supervision implemented by concatenating the T per-iteration outputs along the batch dimension so that the standard PFN loss averages over iterations. Loop-1L has 2.97M parameters and T=12, Loop-2L 5.33M and T=6, and the 1L control 2.97M. The final training losses are 1.21, 1.13 and 1.61.

Training cost. Table 3 summarizes model size and training cost. Every job ran on a single NVIDIA RTX A6000 (48 GB) of a shared eight-GPU server, and training took 7 to 10 GPU-hours per model. Measured side by side on idle GPUs, a training epoch takes 885 s for Loop-1L and 883 s for NodePFN-12L, since both are bound by sampling the prior, and Loop-1L needs 26% less GPU memory (1.08 against 1.46 GB). With the prior batches cached, an epoch of either looped model is about 1% shorter than one of NodePFN-12L (332 s for Loop-1L and 333 s for Loop-2L against 336 s), measured over three rounds on three GPUs in which every model trains once on every GPU. The difference is small because a block applied twelve times costs as much computation as twelve layers, and the saving plausibly comes from updating ten times fewer parameters. The whole study used about 70 GPU-hours of wall-clock time, including development runs, which is an upper bound because other jobs shared the server.

Table 3: Model size and training cost on one A6000. Memory is peak GPU memory in training. An epoch is timed without prior sampling (the prior batches cached), as the mean over three rounds on three GPUs in which every model trains once on every GPU, and end to end with the models trained side by side.

Model	Params	Checkpoint	Memory	Epoch, model only	Epoch, end to end
NodePFN-12L	29.0M	116 MB	1.46 GB	336 s	883 s
Loop-1L	2.97M	12 MB	1.08 GB	332 s	885 s
Loop-2L	5.33M	21 MB	1.10 GB	333 s	883 s

Inference cost. Table 4 gives the inference cost by graph size and Table 5 for every graph, measured on an idle GPU. After ICDS the prediction is faster than NodePFN-12L on 21 of the 23 graphs, by 2.2 times on the median graph and 3.0 times over all 23 graphs, and the speedup grows with the graph, from 1.3 times under 1,000 nodes to 2.7 times over 10,000. A prediction costs as many block applications as the selected depth, so the speedup is set by $\hat{T}$ (Spearman $\rho{=}-0.97$ across the graphs), from 5.1 times on actor ($\hat{T}{=}2$) and about 3.5 times at $\hat{T}{=}3$ to parity at $\hat{T}{=}12$. On dblp, for example, ICDS takes 2.7 s once and raises the accuracy of Loop-1L from 64.0% at the training depth to 71.8%, that of NodePFN-12L, and every later prediction is 3.4 times faster. Every iteration is trained to be decoded (§4.2), so a looped model can also stop before $\hat{T}$ when time is short, while NodePFN-12L can stop early only through a decoder trained for each layer, as in the early-exit readout of §3.2.

One-off cost of ICDS. ICDS runs once per graph and takes 2.7 s on the median graph, so a single prediction with ICDS takes 2.6 s longer than one of NodePFN-12L. On small graphs this overhead is trivial, e.g., on texas ICDS and the prediction together take 0.05 s against 0.01 s for NodePFN-12L, and on the six graphs under 1,000 nodes the overhead is at most 0.23 s. With five hold-outs instead of twenty, ICDS takes 0.7 s on the median graph, and the mean accuracy is 66.0% against 66.3% for Loop-1L and 66.9% with either number of hold-outs for Loop-2L.

Exceptions. The prediction is slower than NodePFN-12L only on air-europe and air-brazil, where ICDS selects more iterations than the training depth. On graphs under 5,000 nodes the twenty hold-outs of ICDS run in one batch to save time, which is why Table 5 lists more GPU memory for the looped models on those graphs (at most 8.4 GB, on squirrel, against 1.0 GB for the prediction). Across the 23 graphs ICDS raises the mean accuracy from 64.6% to 66.3% on this split and lowers it on three graphs, most on air-usa (from 56.2% to 53.7%).

Table 4: Inference cost by graph size on one idle A6000, medians over the graphs of each group. A looped model predicts at the depth $\hat{T}$ selected by ICDS, with its median speedup over NodePFN-12L in parentheses, and ICDS runs once per graph with 20 / 5 hold-outs. Table 5 gives every graph.

	NodePFN-12L	Loop-1L		Loop-2L	
Graphs	Pass (ms)	Pass (ms)	ICDS (s)	Pass (ms)	ICDS (s)
<1,000 nodes (6)	11	10 (1.3×)	0.05 / 0.02	10 (1.3×)	0.07 / 0.03
1,000–10,000 (6)	35	20 (2.4×)	0.66 / 0.17	20 (1.8×)	0.97 / 0.25
>10,000 (11)	186	83 (2.7×)	4.4 / 1.1	174 (1.9×)	6.6 / 1.7
All (23)	106	31 (2.2×)	2.7 / 0.67	67 (1.8×)	4.0 / 1.0

ICDS design and sensitivity. ICDS uses the hold-out fraction $q{=}0.2$, accuracy scoring and 20 hold-outs, and keeps the query nodes in the sequence. Removing them leaves, e.g., 140 of Cora's 2708 nodes in a nearly edgeless graph on which propagation depth is invisible, so this variant selects two or three iterations regardless of h ($|\rho| \leq 0.05$) and without smoothing falls below the training depth (Table 7). Table 6 varies q and the number of hold-outs on all 23 graphs (Loop-1L, split 0). Every setting improves on the training depth. The default setting is the best under the standard protocol and within 0.4% of the best without smoothing, and with five hold-outs instead of twenty

Table 5: Inference cost per graph on one idle A6000 (standard protocol, split 0, preprocessing and checkpoint loading excluded). Pass is one prediction pass, for a looped model at the depth $\hat{T}$ selected by ICDS, which runs once per graph. Pass times are medians of nine rounds in which the three models alternate, GB is peak GPU memory, and bold marks the fastest pass.

			NodePFN-12L, 29.0M		Loop-1L, 2.97M				Loop-2L, 5.33M			
Dataset	Nodes	Context	Pass (ms)	GB	ICDS (s)	$\hat{T}$	Pass (ms)	GB	ICDS (s)	$\hat{T}$	Pass (ms)	GB
Homophilous												
cora	2,708	140	21.8	0.7	0.38	8	15.5	2.1	0.56	1	**5.9**	2.1
citeseer	3,327	120	24.2	0.8	0.44	3	**8.4**	2.3	0.65	3	13.8	2.3
pubmed	19,717	60	111	1.0	2.8	3	**31.5**	1.0	4.2	2	42.2	1.0
air-usa	1,190	80	17.8	1.0	0.28	4	**8.5**	3.3	0.42	4	13.3	3.3
air-europe	399	80	12.6	0.8	0.12	15	14.8	1.8	0.18	4	**9.2**	1.8
air-brazil	131	80	**11.4**	0.9	0.05	16	14.3	1.1	0.06	12	23.0	1.1
wikics	11,701	580	169	2.1	4.2	4	**62.4**	2.1	6.3	6	172	2.1
amazon-computer	13,752	200	186	2.3	4.7	5	**83.3**	2.3	7.0	3	98.7	2.3
amazon-photo	7,650	160	97.4	1.6	2.4	4	**37.0**	1.6	3.6	5	84.1	1.6
dblp	17,716	80	106	1.2	2.7	3	**30.9**	1.2	4.0	7	125	1.2
coauthor-cs	18,333	300	135	1.4	3.3	6	**70.5**	1.4	5.0	3	71.5	1.4
coauthor-physics	34,493	100	274	2.6	7.0	7	**165**	2.6	10.5	5	230	2.6
deezer	28,281	40	170	1.3	4.3	4	**62.4**	1.3	6.4	12	331	1.3
Heterophilous												
cornell	183	87	10.2	1.0	0.05	4	**3.7**	1.1	0.06	3	5.3	1.1
texas	183	87	10.0	0.9	0.05	6	5.2	0.9	0.06	2	**3.6**	0.9
wisconsin	251	120	10.6	0.9	0.06	12	**10.5**	1.0	0.08	6	10.5	1.0
chameleon	890	409	16.5	0.9	0.23	6	**9.9**	2.5	0.34	4	12.4	2.5
actor	7,600	3,648	125	1.1	2.9	2	**24.4**	1.1	4.2	3	66.8	1.1
minesweeper	10,000	5,000	191	1.3	4.4	12	**191**	1.3	6.6	7	222	1.3
tolokers	11,758	5,879	480	4.3	11.1	5	213	4.3	16.7	2	**174**	4.3
amazon-ratings	24,492	12,246	1010	4.9	21.4	4	342	4.9	32.1	2	**342**	4.9
questions	48,921	24,460	4100	18.5	87.0	3	1039	18.5	130.6	1	**697**	18.5
squirrel	2,223	1,053	46.0	1.0	0.88	8	31.4	8.4	1.3	3	**25.3**	8.4
All 23 graphs, total / max			7.3 s	18.5	161		2.5 s	18.5	241		2.8 s	18.5

ICDS reaches 66.0% against 66.3% under the standard protocol and 64.1% against 64.3% without smoothing.

Table 6: ICDS sensitivity (Loop-1L, 23 graphs, split 0). Accuracy in %, gain over the training depth $T{=}12$, number of graphs where ICDS is ahead, Wilcoxon p, and the share of the oracle's gain over the training depth that ICDS recovers.

Condition	q / hold-outs	fixed	ICDS	Δ	wins	p	oracle gain recovered
standard	0.1 / 20	64.6	65.8	+1.2	14	0.024	39%
	0.2 / 5	64.6	66.0	+1.4	16	0.021	44%
	0.2 / 20	64.6	**66.3**	+1.8	17	0.002	55%
	0.2 / 50	64.6	66.3	+1.7	16	0.006	53%
	0.3 / 20	64.6	66.0	+1.4	13	0.035	44%
no smoothing	0.1 / 20	63.6	**64.7**	+1.2	16	0.007	39%
	0.2 / 5	63.4	64.1	+0.7	12	0.29	23%
	0.2 / 20	63.4	64.3	+1.0	12	0.062	33%
	0.2 / 50	63.4	64.6	+1.3	14	0.015	46%
	0.3 / 20	63.5	64.6	+1.1	11	0.085	38%

When ICDS fails: label-sparse graphs. With harder features ($\mu{=}0.5$) and only 5% of the nodes labeled (51 nodes, hence 10 pseudo-queries per hold-out, on 120 cSBM graphs), ICDS selects $\hat{T}{=}1$ on 37 of 120 graphs and reaches 62.1% against 63.7% at the training depth (Wilcoxon $p{<}10^{-4}$), while the oracle reaches 65.2%. The cause is the held-out curve, which overestimates the first iteration's accuracy by 7%. With ten held-out nodes the curve is too coarse to locate an optimum, so with features this weak ICDS needs more than roughly fifty labeled nodes.

Depth rules. Table 7 compares seven ways of choosing the depth for both looped models on split 0. They are the training depth, the best single global T for all graphs chosen with test labels (one

integer, so a weak oracle), ICDS, ICDS on the context-only graph (§4.3), ICDS choosing the branch schedule of §5 together with T, the per-graph oracle, and the oracle over (T, g). A global T recovers about half of what ICDS does under the standard protocol and slightly more without smoothing, where the best global T is 7–8 rather than 12. Only the per-graph rules follow homophily, and only when the query nodes stay in the sequence.

Table 7: Depth rules (split 0). Mean accuracy over the 23 graphs in % (in parentheses, the difference to NodePFN-12L's 67.6 standard / 65.8 no smoothing). The last two columns give $\rho(\hat{T}, h)$ for Loop-1L and Loop-2L. With schedule: message passing in the first g iterations only, (T, g) chosen together (§5).

Condition	Rule	Loop-1L×12	Loop-2L×6	$\rho(\hat{T}, h)$ 1L	$\rho(\hat{T}, h)$ 2L
Standard (NodePFN k)	fixed, trained T (12 / 6)	64.6 (-3.0)	65.6 (-2.0)	–	–
	fixed, best global T (7 / 3)	65.9 (-1.6)	66.4 (-1.2)	–	–
	ICDS	66.3 (-1.3)	66.9 (-0.7)	-0.17	+0.17
	ICDS, ctx-only graph	65.5 (-2.1)	66.3 (-1.3)	+0.11	-0.52
	ICDS, with schedule	66.1 (-1.5)	67.3 (-0.3)	-0.20	-0.32
	oracle	67.7 (+0.2)	68.5 (+0.9)	-0.09	+0.00
	oracle, with schedule	69.1 (+1.5)	69.3 (+1.7)	-0.59	-0.12
No smoothing (k=0)	fixed, trained T (12 / 6)	63.4 (-2.4)	64.1 (-1.7)	–	–
	fixed, best global T (7 / 8)	64.7 (-1.1)	64.3 (-1.6)	–	–
	ICDS	64.3 (-1.5)	64.4 (-1.4)	+0.43	+0.63
	ICDS, ctx-only graph	63.2 (-2.6)	61.7 (-4.1)	-0.05	-0.04
	ICDS, with schedule	64.9 (-0.9)	64.8 (-1.0)	+0.48	+0.47
	oracle	66.4 (+0.6)	66.1 (+0.3)	+0.72	+0.47
	oracle, with schedule	67.3 (+1.5)	66.9 (+1.1)	+0.33	+0.32

Per-node depth. ICDS picks one depth per graph. From the per-node prediction trajectories (23 graphs, both conditions, both looped models) we evaluated three label-free per-node rules. A node stops once its predicted class has been stable for $s \in \{2, 3\}$ consecutive iterations, or at its most confident iteration up to $\hat{T}$. None reaches the per-graph choice. The best (s=3) is 0.3–1.6% below ICDS and s=2 is 1.3–6.3% below. A better per-node rule could still help, since a per-node oracle, which credits a node when any iteration predicts it correctly, reaches 80–82%, 13–18% above ICDS, but a node's own trajectory does not reveal when it is right. Exploiting it would need a learned router, as in mixture-of-recursions models (Bae et al., 2025).

Branch schedule. Table 8 gives the numbers behind the last paragraph of §5. It reports accuracy as all, homophilous and heterophilous means over the 23 graphs (mean ± std over three splits) when the message-passing branch is kept for the first $g \in \{1, 2, 3, \text{all}\}$ iterations and zeroed in later iterations, with (T, g) chosen by ICDS or by an oracle. The oracle prefers a finite g on 8–17 of the 23 graphs, ICDS on 5–12.

Other models and conditions. Figure 13 repeats Fig. 8 for Loop-1L and Loop-2L under the standard protocol and for Loop-1L without smoothing (Loop-2L without smoothing is Fig. 8b). Without smoothing the chosen depth rises with h. On split 0 this falls short of significance for Loop-1L (ρ=0.43, p=0.067), and pooled over graph–split pairs it is significant for both models. For Loop-2L the oracle depth rises with h (ρ=0.42, $p \leq 10^{-3}$) and the ICDS depth follows it (ρ=0.31, p=0.017). For Loop-1L the oracle depth rises from 6.3 iterations on heterophilous to 9.3 on homophilous graphs (ρ=0.46, $p \leq 10^{-3}$) and the ICDS depth from 7.0 to 8.7 (ρ=0.36, p=0.007).

Every iteration smooths. Figure 14 gives the Dirichlet energy per iteration for all four model–condition pairs, the per-iteration counterpart of the per-layer energy of §3.1 (Fig. 4b). In NodePFN-12L the energy drops in layers 1–2 and then rises again (without smoothing, heterophilous group mean $0.68 \to 0.32 \to 0.42$ by layer 5), because its later layers propagate little and their attention and feed-forward blocks re-sharpen the representation. In the looped models it falls at every iteration ($0.43 \to 0.27$ for Loop-1L) and never recovers, because the shared block propagates each time. This is consistent with the decline past the optimum and the lower accuracy on heterophilous graphs, where ICDS stops earlier than on homophilous graphs while the energy is still falling.

Table 8: In-context branch schedule, mean ± std over three splits. Accuracy (%) over all 23 graphs, the 13 homophilous and the 10 heterophilous ones, when message passing is kept for the first g iterations only, with (T, g) chosen by ICDS or by an oracle, and `NodePFN-12L` for reference.

Condition	Model	Graphs	NodePFN-12L	ICDS	ICDS+sched.	Oracle T	Oracle T+sched.
Standard	Loop-1L	all	66.6±0.7	66.3±0.3	66.2±0.1	67.8±0.2	69.0±0.0
		homo.	68.5±1.0	69.3±0.3	68.5±0.8	70.6±0.5	71.6±0.5
		hetero.	64.2±0.4	62.3±0.7	63.3±0.9	64.1±0.3	65.7±0.6
	Loop-2L	all	66.6±0.7	66.5±0.3	67.0±0.2	68.3±0.2	69.4±0.2
		homo.	68.5±1.0	70.1±0.7	70.2±0.7	72.0±0.6	72.6±0.5
		hetero.	64.2±0.4	61.8±0.4	63.0±0.7	63.5±0.2	65.2±0.7
No smoothing	Loop-1L	all	65.0±0.6	64.5±0.5	64.9±0.4	66.3±0.3	67.2±0.5
		homo.	66.0±0.7	66.3±0.3	66.4±0.2	68.0±0.3	68.4±0.3
		hetero.	63.7±0.5	62.1±0.9	63.1±1.0	64.0±0.3	65.5±0.8
	Loop-2L	all	65.0±0.6	64.6±0.4	65.3±0.4	66.2±0.0	67.3±0.3
		homo.	66.0±0.7	67.0±0.3	67.1±0.2	68.5±0.1	69.0±0.1
		hetero.	63.7±0.5	61.6±0.5	62.8±0.8	63.3±0.2	65.0±0.7

cSBM and syn-Cora with `Loop-2L`. Figure 15 (d–f) repeats the cSBM and syn-Cora experiments of Fig. 8c, d with `Loop-2L`×6 ($T_{\max}$=12, one iteration is two propagation steps). On syn-Cora `Loop-2L` behaves like `Loop-1L`. The oracle and ICDS depth are about 2 for every $h < 0.5$ and rise to 7 at h=0.95 (ρ=0.97 and 0.98 with h), and ICDS raises accuracy from 62.9% to 65.7% (from 50.1% to 55.4% on the heterophilous half, 23 of 30 graphs, $p<10^{-4}$). On the two-class cSBM the block's two hops per iteration make the depth–κ relation flatter (Fig. 15d).

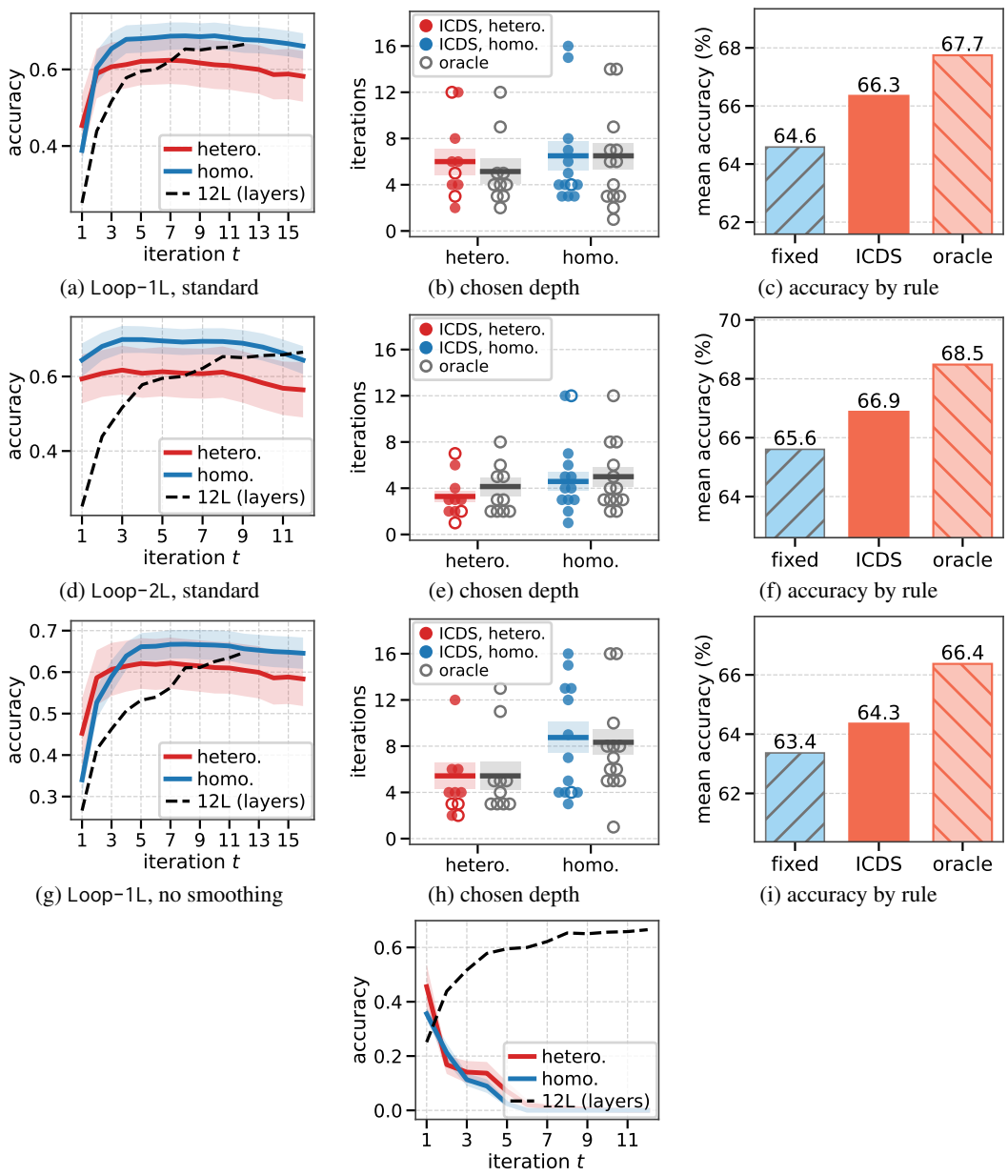
F Related work in detail

Mechanistic studies of depth. Logit and tuned lenses (nostalgebraist, 2020; Belrose et al., 2023) and studies of layer skipping and stages of inference in language models (Sun et al., 2025; Lad et al., 2025) analyze depth layer by layer. We follow the protocol that Balef et al. (2026) built for tabular foundation models and add what the graph setting needs. Switching off one branch of one layer separates refinement from propagation, and the Dirichlet energy and a homophily probe have no tabular analogue. Their finding that middle layers are redundant holds here for refinement only. Propagation is needed, and its useful amount is determined by the graph, whereas their loop count is fixed at six. We also find that a tuned evaluation pipeline can hide this dependence (§3.2). We are not aware of a comparable study of a graph foundation model.

Looped and recurrent transformers. Weight-tied depth goes back to Universal Transformers (Dehghani et al., 2019), and looped transformers (Giannou et al., 2023; Saunshi et al., 2025) iterate a small block. Recurrent-depth LMs (Geiping et al., 2025), HRM and TRM (Wang et al., 2025; Jolicoeur-Martineau, 2025), elastic-depth LMs (Jeddi et al., 2026) and mixture-of-recursions (Bae et al., 2025) all train a small block to be iterated and vary the number of iterations at test time, by input difficulty or by a compute budget. Our per-iteration supervision resembles the deep supervision of HRM and TRM. A one-layer model trained without looping collapses when looped at inference (§5).

What an iteration is. In those models an iteration is pure refinement, so accuracy rises with test-time iterations until it saturates (Geiping et al., 2025). In a PFN for graphs every iteration is also a hop of propagation, the useful number of hops is bounded by the graph (Proposition 1, finite for every κ and zero below κ_c), and accuracy *falls* past the optimum (Fig. 8a). Test-time depth therefore needs to be selected per graph instead of being scaled up.

Selecting the depth. Adaptive-depth GNNs (Chien et al., 2021; Spinelli et al., 2021; Hevapathige et al., 2026) learn how far to propagate by training on the target graph, and halting heads, post-hoc readouts (Popescu et al., 2026) and per-token recursion learn when to stop from training data. ACT-style halting heads (Graves, 2016; Banino et al., 2021; Wang et al., 2025) are trained on the prior,



(a) Loop-1L, standard (b) chosen depth (c) accuracy by rule

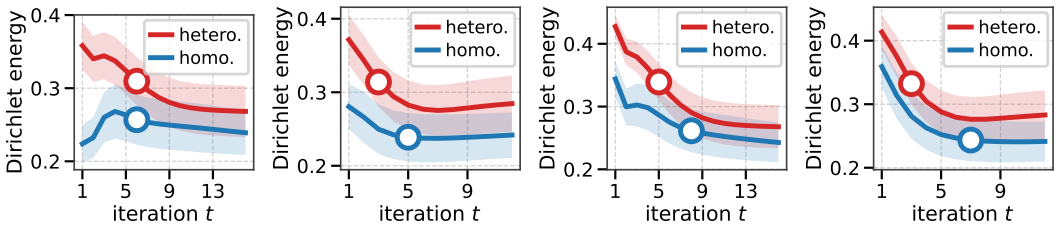
(d) Loop-2L, standard (e) chosen depth (f) accuracy by rule

(g) Loop-1L, no smoothing (h) chosen depth (i) accuracy by rule

(j) NodePFN-1L looped, standard

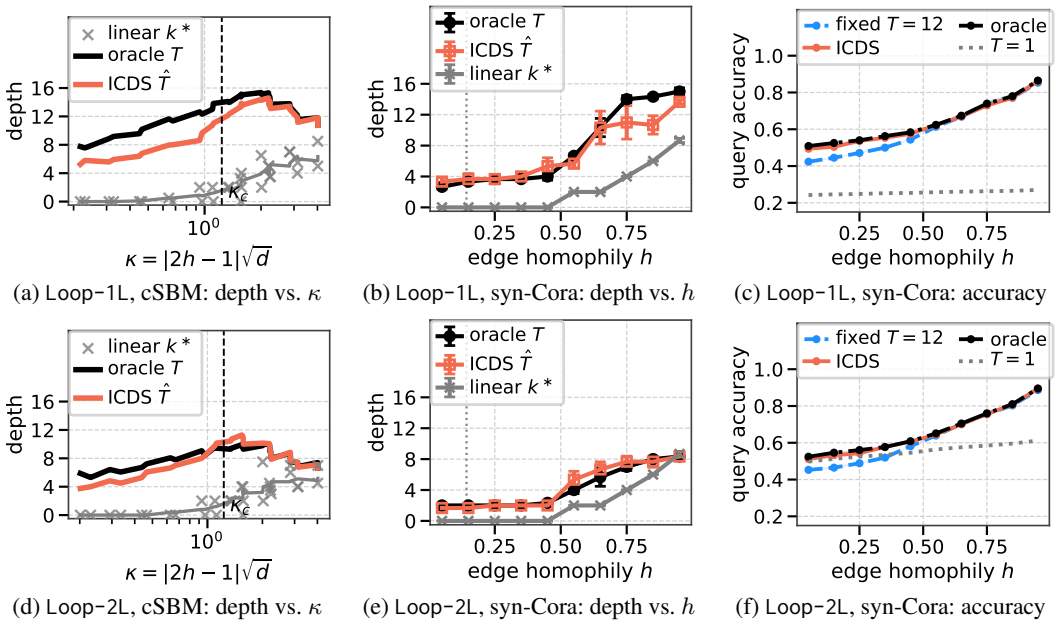
Figure 13: Looped NodePFN, other models and conditions. Accuracy per iteration and chosen depth as in Fig. 8a, b, and mean accuracy by rule for Loop-1L under the standard protocol (a–c), Loop-2L under the standard protocol (d–f) and Loop-1L without smoothing (g–i). The chosen depth is unrelated to h under the standard protocol and rises with it without smoothing ($\rho = -0.17$, 0.17 and 0.43, $p = 0.48$, 0.48 and 0.067). (j) NodePFN-1L, trained without looping, applied repeatedly at inference (split 0): accuracy falls below 1% by $t = 6$ in both groups.

whereas ICDS is training-free because a PFN has access to labeled context at test time. Per-token recursion (Bae et al., 2025) is the learned analogue of the per-node rules that we evaluate without labels in Appendix E. Latent chain-of-thought for tabular PFNs (Dudley & Oymak, 2026) loops for refinement only. Our contribution is the finding that the loop's useful length is a graph property, a gradient-free rule that reads that property from the labeled context, and the observation that a tuned pipeline can hide it.



(a) Loop-1L, standard (b) Loop-2L, standard (c) Loop-1L, no smoothing (d) Loop-2L, no smoothing

Figure 14: **Every iteration smooths.** Dirichlet energy of the query representations after each iteration, group mean ± s.e., for both looped models under both conditions. The open circle marks the mean depth ICDS selects for each group. Unlike NodePFN-12L, whose energy rebounds after layer 2 (Fig. 4b), the looped block lowers it at every iteration.



(a) Loop-1L, cSBM: depth vs. κ (b) Loop-1L, syn-Cora: depth vs. h (c) Loop-1L, syn-Cora: accuracy

(d) Loop-2L, cSBM: depth vs. κ (e) Loop-2L, syn-Cora: depth vs. h (f) Loop-2L, syn-Cora: accuracy

Figure 15: Proposition 1 on the looped models, Loop-1L (a–c) and Loop-2L (d–f, $T_{\max}$=12, one iteration is two propagation steps). Depth panels: linear depth k^* (×), oracle and ICDS iterations. Accuracy panels: after one iteration, at the training depth, with ICDS and with the oracle.

Scope. Our claims concern node classification with PFNs for graphs in the single-pass setting. The mechanistic results are established on NodePFN and reproduced on GraphPFN, which we do not loop because its backbone is frozen. We reproduce our lens and branch ablations on GraphPFN (Appendix C), but looping it would require training the backbone block to be iterated, beyond the adapter-only pretraining the authors report (≈290 A100-hours), and we leave that to future work.

G Threats to validity and limitations

This section expands the closing paragraphs of §7.

Threats to validity. The depth ICDS selects follows homophily only without pre-smoothing, where the correlation is significant over the 57 informative graph–split pairs (ρ=0.36 for Loop-1L and 0.31 for Loop-2L, Appendix E). Our conclusions on prediction depth rest on ℓ^*, whose growth with homophily was hypothesized before the experiments, and ℓ_{rep} serves as a descriptive complement. ICDS uses only the labels of the context nodes, and test labels enter only the oracle rules, which Table 1 shows in gray without ranking. Every setting of q and of the number of hold-outs in

Table 6 raises the mean accuracy over the training depth, so the gain does not rest on one setting. Proposition 1 covers the two-class cSBM. For several classes it is supported by a numerical check and by syn-Cora, on which the selected depth rises with h (Appendix D, Fig. 8d). Every accuracy in this paper, in the analysis of §3 and in the experiments of §5 alike, comes from a single pass with one ensemble member, so that each comparison reflects the network alone. This places `NodePFN-12L` below the ensembled numbers reported by Choi et al. (2026), as detailed in Appendix A.

Limitations. ICDS runs once per graph and takes a few seconds on most graphs (2.7 s on the median graph, Table 5), and fewer hold-outs shorten it further, to about a quarter with five (Appendix E, One-off cost of ICDS). The cost of ICDS and that of a prediction grow with the size of the graph. On the largest benchmark, questions with 48,921 nodes, ICDS takes 87 s once, and a prediction then takes 1.0 s for `Loop-1L` against 4.1 s for `NodePFN-12L` (Table 5). With weak features and fewer than about fifty labeled nodes, ICDS falls below the training depth, because the held-out curve overestimates the accuracy of the first iteration (Appendix E, "When ICDS fails"). ICDS selects one depth per graph, and the label-free per-node rules we evaluated are less accurate than this single depth (Appendix E, "Per-node depth"). Choosing the branch schedule in context recovers only part of the oracle's gain (Table 1, Table 8). On heterophilous graphs the looped models remain below `NodePFN-12L` under every in-context rule (Table 1), which is consistent with the smoothing that each iteration adds (Appendix E, "Every iteration smooths"). With the branch schedule chosen by ICDS, `Loop-1L` reaches 63.3% on these graphs under the standard protocol, against 62.3% without it and 64.2% for `NodePFN-12L`.