Efficient Dynamic Algorithms for Graph Neural Networks with Non-Linear Propagation

Kiarash Banihashem[*]
University of Maryland, College Park
kiarash@umd.edu

MohammadTaghi Hajiaghayi[*]
University of Maryland, College Park
hajiagha@umd.edu

Mahdi JafariRaviz[*]
University of Maryland, College Park
mahdij@umd.edu

Silvio Lattanzi[*]
Google Research
silviol@google.com

Danny Mittal[*]
University of Maryland, College Park
mittal@umd.edu

Abstract

Graph Neural Networks (GNNs) are widely used for representation learning on graphs, but most methods assume static topologies, making them inefficient on evolving networks where edges change over time. Existing dynamic approaches either model graph evolution through temporal GNN architectures without focusing on efficient dynamic maintenance, or are restricted to linear propagation models based on Personalized PageRank.

In this work, we study how to efficiently maintain node representations for non-linear GNN propagation under edge insertions and deletions. The propagation has no learned parameters, and only a classifier applied afterward is trained. For a broad class of standard activation functions, we develop a residual-based dynamic algorithm that selectively propagates local errors via push operations, maintaining an approximation to the evolving fixed point without full recomputation. We prove that our method achieves amortized $O(1/\epsilon)$ update time per graph change under a degree-normalized error guarantee. Our approach uses a potential-based analysis in a degree-scaled norm and, in contrast to prior work on the linear case, requires no randomness assumptions on either the update sequence or the input vector. For the linear special case, we additionally provide an exact dynamic algorithm via low-rank matrix inverse updates. Experiments on benchmark datasets show that incorporating non-linearity improves accuracy while preserving efficient update performance, yielding a scalable and theoretically grounded method for maintaining this propagation on dynamic graphs.

1 Introduction

Graph Neural Networks (GNNs) are widely used in machine learning, both for prediction tasks such as node classification and as general-purpose tools for representation learning [HYL17; KW17]. Their ability to exploit the underlying graph structure of real-world data makes them effective in many domains such as biology, social sciences, and transportation [Wu+20]. A key challenge, however, is that real-world data is often *dynamic*: the relationships between the nodes of the graph are constantly

[*]Equal contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

changing, requiring the model to update its outputs accordingly. A naive way of satisfying this requirement is to periodically recompute the model after each change in order to incorporate changes in the graph. However, such recomputation is computationally expensive and therefore undesirable both theoretically and practically.

Existing approaches to dynamic GNNs can be broadly divided into two lines of work. The first line is *temporal* GNNs which model graph evolution using recurrent architectures, attention mechanisms, or memory modules [KZL18; Tri+19; Xu+20; Ros+20]. These approaches are primarily geared toward prediction tasks such as node classification and are not designed for efficiently maintaining representations under graph changes [Zhe+22; ZWL23]. The second line focuses on the propagation component of GNNs and considers *linear* GNNs where each node has a value obtained via iterated neighborhood averaging [Zhe+22; ZWL23]. In this setting, the model reduces to *Personalized PageRank (PPR)*, where the structure enables efficient updates, allowing the representation vectors to be maintained under graph changes without recomputing from scratch [BCG10; ZLG16].

A natural question is whether one can efficiently maintain the representations of *non-linear* GNNs under graph updates. Non-linearity is central to the success of GNNs, as it enables the model to capture complex patterns in the data. By repeatedly composing non-linear activations with neighborhood aggregation, GNNs gain expressive power analogous to deep neural networks in standard (non-graph) settings.

Our Contributions. In this paper, we answer the above question affirmatively. We study GNNs in which each node applies a non-linear activation to a weighted average of its neighbors. The propagation has no learned parameters, and its fixed point is unique. Training happens only in a classifier applied to the propagated features, as in InstantGNN [Zhe+22]. We assume the activation function is K-Lipschitz for some $K \leq 1$, a condition that is sufficient for convergence. This restriction is mild in the GNN setting. Common activations such as ReLU, tanh, sigmoid, hard tanh, softplus, softsign, and ELU are 1-Lipschitz or smaller. Under this assumption, we show that one can maintain an approximate solution for all nodes in amortized time $O(1/\epsilon)$, where ϵ is the degree-normalized approximation parameter. More precisely, the amortized time is $O\big((1+B_z)/(1-K(1-\alpha))^2 \cdot (1/\epsilon)\big)$, where α is the teleportation probability and B_z bounds the size of the fixed point (Section 2.1). For $K = 1$, as for ReLU and tanh, the factor $1 - K(1-\alpha)$ equals α. In addition, for the linear special case, we provide an exact dynamic maintenance algorithm based on low-rank matrix-inverse updates, obtaining a worst-case update time $O(n^{1.5275})$, where n is the number of nodes. We complement our theoretical results with empirical evidence demonstrating the importance of non-linearity for achieving high accuracy. On large dynamic graph benchmarks, we show the practical behavior of our proposed approximation algorithm.

Our Techniques. To maintain an approximate solution under dynamic updates, we use a residual vector, as in prior work [Zhe+22; ZLG16]. We use a push operation to propagate large residuals, and we show that when all residuals are small relative to the degree-scaled error, the maintained solution is a valid approximation. For an edge update, two effects must be handled: (i) the new values that should be propagated through the updated edge, and (ii) the effect of the degree changes at the two endpoints on their neighbors, which arises from the edge-weight normalization. The first effect can be handled in $O(1)$ time using the scaling trick of Zhang, Lofgren, and Goel [ZLG16], which is not part of the main algorithm of Zheng et al. [Zhe+22]. After applying the effect of the edge update, the residuals at the endpoints may increase, and we propagate them when needed. The main difference from prior work is in the analysis of this cleanup step. Our setting requires handling the nonlinear update rule, and we also remove the randomness assumption used in prior work. We remove this assumption and obtain a deterministic guarantee, at the cost of a weaker amortized update bound.

Related Work A line of work in dynamic GNNs studies temporal graphs, where the primary focus is on prediction tasks over evolving graphs [KZL18; Tri+19; Xu+20; Ros+20]. These approaches are typically divided into *discrete-time dynamic* (DTD) and *continuous-time dynamic* (CTD) models: the former takes as input a sequence of graph snapshots at different time steps, while the latter operates on a stream of timestamped events that modify the graph. We refer to [ZYW25] for a comprehensive overview. These models use the timing and order of graph changes as a signal for prediction, and some of them, such as Rossi et al. [Ros+20], keep a memory for each node that depends on past events. In contrast to this line of work, our focus is not on prediction accuracy but on *efficiently maintaining* node embeddings under graph updates with *theoretical guarantees*. In particular, we

provide amortized update time guarantees for our algorithm. The embeddings we maintain depend only on the current graph, so our method fits tasks where the history of changes does not matter, while temporal GNNs fit tasks where it does.

The closest line of work to ours is the literature on dynamic *Personalized PageRank* (PPR), which can be viewed as a special case of our model with update function $f_i(x) = x$. For this problem, [ZLG16] build on the Push framework of [ACL06] and give a dynamic algorithm that maintains the PageRank solution for a randomly chosen source node. In our notation, this corresponds to setting s to a random basis vector. Their algorithm has amortized update time $O(1/(n\epsilon))$ when $\beta \in \{0, 1\}$, where β is the normalization parameter in Equation (1). Building on this line of work, [Zhe+22] study a linear GNN model and obtain the same $O(1/(n\epsilon))$ update time for general β, but under an additional randomness assumption on the edge updates: each edge is assumed to be equally likely to be inserted at each time. In contrast to these works, we allow nonlinear updates and make no randomness assumptions on either the update sequence or the vector s.

A separate line of work on dynamic PageRank uses random walk sampling to maintain the solution vector [BCG10; Bah+12; Jay+24]. On a high level, this approach estimates each node's solution value by averaging over a collection of random walks starting from the node. When an edge is inserted or deleted, only the affected walks are re-sampled in order to maintain correctness. This line of work was initiated by [BCG10], who analyzed the setting under random edge updates. More recently, [Jay+24] extend this approach to general (adversarial) updates and obtain upper and lower bounds for both additive ℓ_1 error and multiplicative error. The sampling technique heavily relies on the linearity of PageRank, which allows a node's solution to be computed independently of its neighbors' solutions. As such, this approach does not extend to the nonlinear setting considered in our work.

Relation to other GNN models. As in the GNN of Scarselli et al. [Sca+08], our node values are the fixed point of a contraction, but our update function is fixed rather than learned. Our model is closest to InstantGNN [Zhe+22], which also trains only a classifier on the propagated features, and to APPNP [GBG18], whose propagation also has no learned parameters. Unlike the most popular GNNs, our propagation has no learned weights, no mixing across feature dimensions, and no attention. Maintaining a version with a learned matrix that mixes the features is left open.

2 Dynamic Non-linearity

2.1 Preliminaries

To model nonlinearity, we consider an undirected, unweighted graph $G = (V, E)$ with permanent self-loops at every node. We assume $V = \{1, \ldots, n\}$, where n is the number of nodes. Table 2 in the Appendix lists all the parameters used in the paper and their roles. Let A be the adjacency matrix of this graph. We define the normalized weight matrix W as

$$W = D^{-\beta} A D^{\beta-1}, \tag{1}$$

where $0 \leq \beta \leq 1$ is the normalization parameter, and D is the diagonal degree matrix. [2] We denote the degree of node i by $d(i)$. Since every node has a self-loop, $d(i) \geq 1$ for all i, W is well-defined.

Given a vector $s = (s_1, \ldots, s_n)^T \in \mathbb{R}^n$ and teleportation probability α, we consider the following dynamics. Let $z(t) = (z_1(t), \ldots, z_n(t))^T$ be the vector of node values at time t. We define $z(t+1)$ from $z(t)$ by

$$z_i(t+1) = f_i\left(\alpha s_i + (1-\alpha)\sum_j w_{ij} z_j(t)\right), \tag{2}$$

for every $i \in V$, where f_i is a node-wise function. In vector form, this update is

$$z(t+1) = F\left(\alpha s + (1-\alpha) W z(t)\right), \tag{3}$$

where $F\colon \mathbb{R}^n \to \mathbb{R}^n$ applies the functions f_i pointwise. The vector s holds the input values. We also define the induced map $T(z) = F\left(\alpha s + (1-\alpha) W z\right)$. When F is the identity map, this recovers the linear model considered by [Zhe+22].

[2] We adopt this normalization as it is standard in related literature. Any matrix W with a spectral radius $\rho(W) \leq 1$ imposes a similar convergent dynamic with a unique fixed point, though the dynamic algorithm may vary.

Throughout the paper, we use the following assumptions:

- $\alpha \in (0, 1)$, and $\beta \in [0, 1]$;
- each f_i is K-Lipschitz with $K \leq 1$, that is, for all $x, y \in \mathbb{R}$, $|f_i(x) - f_i(y)| \leq K|x - y|$;
- there is a constant B_z such that $\|D^{\beta-1}z^*\|_\infty \leq B_z$ for the fixed point z^* of T on any graph.
  - If each f_i takes values in $[-B_z, B_z]$, the condition holds (Theorem A.1 in the Appendix).
  - For any activation, it holds with $B_z = (\max_i |f_i(0)| + K\alpha\|s\|_\infty)/(1 - K(1 - \alpha))$ (Theorem A.3 in the Appendix). For the identity and ReLU, this gives $B_z = \|s\|_\infty$.

We will use the degree-scaled infinity norm, defined as $\|v\|_{D,\infty} = \|D^{\beta-1}v\|_\infty$. The degree-scaled infinity norm satisfies the property in the following lemma, which we use later. Using this norm and the assumptions above, we then show that the fixed point z^* is unique by the Banach fixed-point theorem [Ban22].

Lemma 2.1. *For any vector v, $\|Wv\|_{D,\infty} \leq \|v\|_{D,\infty}$.*

Lemma 2.2. *Assume that $\alpha \in (0, 1)$, $\beta \in [0, 1]$. Further assume that each f_i is K-Lipschitz with $K \leq 1$. Then there is a unique fixed point z^* satisfying $T(z^*) = z^*$. Moreover, the iteration $z(t + 1) = T(z(t))$ converges to z^* for any initial point $z(0)$ as $t \to \infty$.*

The proofs of these lemmas are provided in Appendix A.

2.2 The State

Our goal is to maintain an approximation of the unique fixed point z^* in an evolving graph. The idea is to keep an estimate z of z^*, together with a record of how far z is from being a fixed point. To do this, we maintain three vectors z, y, r, which satisfy the following invariants:

Invariant 1. $y = \alpha s + (1 - \alpha)Wz$.

Invariant 2. $z + r = F(y)$.

Under these invariants, z is our approximation of z^*, and r is the residual vector containing changes that have not yet been propagated. The vector y stores the averaged neighbor values after teleportation. It can be recomputed from z and the current graph, but we maintain it explicitly to support fast graph updates later. If node i were updated once, its value would change from z_i to $f_i(y_i) = z_i + r_i$. So r_i measures how much node i would still change, and $r = 0$ exactly when $z = z^*$.

The next lemma shows that a small residual implies that z is close to z^*.

Lemma 2.3. *Let (z, y, r) satisfy Invariants 1 and 2 and, for all i, $|r_i| \leq (1 - K(1 - \alpha))\epsilon d(i)^{1-\beta}$. Then, for all i, $|z_i - z_i^*| \leq \epsilon d(i)^{1-\beta}$.*

To prove this lemma, we use the following result, whose proof is given in Appendix A. The proof uses that T shrinks distances by the factor $K(1 - \alpha)$ in the degree-scaled infinity norm, and that $z + r = T(z)$ by Invariants 1 and 2.

Lemma 2.4. *Let (z, y, r) satisfy Invariants 1 and 2. Then, for every i, $|z_i - z_i^*| \leq \frac{d(i)^{1-\beta}}{1-K(1-\alpha)}\|D^{\beta-1}r\|_\infty$.*

Proof of Theorem 2.3. The assumption implies $\|D^{\beta-1}r\|_\infty \leq (1 - K(1 - \alpha))\epsilon$. The claim now follows from Theorem 2.4. □

2.3 Residual Propagation

In this section, we define the push operator. This operator moves the residual r_i into z_i, which reduces the residual at node i.

The push operator is shown in Algorithm 1. It takes as input a node i, the graph G, and a state (z, y, r) that is assumed to satisfy Invariants 1 and 2. It returns an updated state (z, y, r) that also satisfies Invariants 1 and 2. The operator runs in $O(d(i))$ time.

Algorithm 1 PUSH(i, G, z, y, r)

Input: Node i, graph G, state (z, y, r)
Output: Updated state (z, y, r)

$\Delta \leftarrow r_i$
$z_i \leftarrow z_i + \Delta$
for each $j \in N(i)$ **do**
  $y_j \leftarrow y_j + (1-\alpha) w_{ji} \Delta$
  $r_j \leftarrow f_j(y_j) - z_j$
end for
$r_i \leftarrow f_i(y_i) - z_i$
return (z, y, r)

We prove the following lemma for this procedure.

Lemma 2.5. *Suppose* PUSH(i, G, z, y, r) *is called on a state* (z, y, r) *that satisfies Invariants 1 and 2, and let* (z', y', r') *be the new state. Then* (z', y', r') *also satisfies Invariants 1 and 2. Moreover, for all* $j \neq i$, $|r'_j| \leq |r_j| + K(1-\alpha) w_{ji} |r_i|$. *Furthermore,* $|r'_i| \leq K(1-\alpha) w_{ii} |r_i|$.

The proof is given in Appendix A. Combining the push operator with Theorem 2.3 gives a simple algorithm for computing an approximation z: repeatedly push any node i whose residual exceeds the bound in Theorem 2.3. When this process terminates, z is a valid approximation of z^*. Algorithm 2 gives the full procedure.

Algorithm 2 CLEANUP(G, z, y, r)

Input: Graph G, state (z, y, r)
Output: Updated state (z, y, r) satisfying $|z_i - z_i^*| \leq \epsilon d(i)^{1-\beta}$ for all i

while there exists i such that $|r_i| > (1 - K(1-\alpha))\epsilon\, d(i)^{1-\beta}$ **do**
  $(z, y, r) \leftarrow$ PUSH(i, G, z, y, r)
end while
return (z, y, r)

2.4 Graph Updates

In this section, we give an algorithm to maintain a state (z, y, r) satisfying invariants 1 and 2 after a single edge insertion or deletion.

Suppose an edge (u, v) is inserted. Since the degrees of u and v change, the invariants are immediately violated for each neighbor of the two endpoints: the degree of each endpoint changes, and so do the normalized edge weights from their neighbors to them. One approach is to iterate over all such neighbors and update their residuals. This is the approach used by [Zhe+22] in Algorithm 2. However, this takes time at least $\Omega(d(u) + d(v))$, which prevents efficient dynamic updates. To avoid this cost, we use a scaling trick introduced by [ZLG16]. Define

$$z'_u = \left(\frac{d(u)+1}{d(u)}\right)^{1-\beta} z_u, \qquad z'_v = \left(\frac{d(v)+1}{d(v)}\right)^{1-\beta} z_v,$$

and set $z'_w = z_w$ for all other $w \notin \{u, v\}$. Then, for the new state (z', y', r'), Invariant 1 continues to hold at every node $w \notin \{u, v\}$ if we keep $y'_w = y_w$. Since $z'_w = z_w$ for such nodes, Invariant 2 also continues to hold there if we keep $r'_w = r_w$. It remains to define y'_u, y'_v and r'_u, r'_v so that Invariants 1 and 2 hold at the two endpoints. We show that this can be done in $O(1)$ time, so the full state can be updated in constant time while preserving Invariants 1 and 2. Algorithm 3 gives the details.

Algorithm 3 UPDATEEDGE($G, u, v, type, z, y, r$)

Input: Graph G, edge endpoints u, v, operation $type$ (INSERT or DELETE), state vectors (z, y, r)
Output: Updated state vectors (z, y, r)

Determine edge operation sign
$\sigma \leftarrow +1$ **if** $type =$ INSERT **else** -1
# Compute intermediate values based on current state
for each endpoint $i \in \{u, v\}$ **do**
  $d_i \leftarrow d(i)$ # Store current degree
  $x_i \leftarrow z_i / d_i^{1-\beta}$
  $S_i \leftarrow \frac{d_i^\beta}{1-\alpha}(y_i - \alpha s_i)$
end for
# Apply updates
for each endpoint $i \in \{u, v\}$ with opposite endpoint $j \in \{v, u\}$ **do**
  $d_i' \leftarrow d_i + \sigma$ # Compute new degree
  $z_i \leftarrow z_i \cdot \left(\frac{d_i'}{d_i}\right)^{1-\beta}$ # Apply the scaling trick
  $y_i \leftarrow \alpha s_i + (1-\alpha)\frac{S_i + \sigma x_j}{(d_i')^\beta}$ # Add the new neighbor j's contribution
  $r_i \leftarrow f_i(y_i) - z_i$ # Update residual state
end for
return (z, y, r)

We end this section by noting that UPDATEEDGE preserves Invariants 1 and 2. We formally show this in Theorem A.7 in the Appendix.

2.5 Dynamic Algorithm

Using the procedures defined above, we now give the dynamic algorithm for our non-linear model. We start from an initial state $(z_0^{\text{start}}, y_0^{\text{start}}, r_0^{\text{start}})$ satisfying Invariants 1 and 2, defined by $z_0^{\text{start}} := 0$, $y_0^{\text{start}} := \alpha s$, and $r_0^{\text{start}} := F(\alpha s)$. We then apply CLEANUP to obtain the post-initial state (z_0, y_0, r_0), whose residuals are small enough to give the desired approximation guarantee. After this initialization step, each edge update is handled by first applying UPDATEEDGE and then applying CLEANUP. Algorithm 4 gives the full procedure.

Algorithm 4 Dynamic Graph Update

Input: Initial graph G_0, edge update events $\{\text{Event}_1, \ldots, \text{Event}_k\}$
Output: Approximation z_t of the fixed point after each update $t = 1, \ldots, k$, satisfying $|z_{t,i} - z_{t,i}^*| \leq \epsilon\, d_t(i)^{1-\beta}$ for all i

$z_0^{\text{start}} \leftarrow 0, \quad y_0^{\text{start}} \leftarrow \alpha s, \quad r_0^{\text{start}} \leftarrow F(\alpha s)$
$(z_0, y_0, r_0) \leftarrow$ CLEANUP$(G_0, z_0^{\text{start}}, y_0^{\text{start}}, r_0^{\text{start}})$
for each $\text{Event}_t = ((u, v), type_t)$ for $t \in \{1, \ldots, k\}$ **do**
  $(z_t', y_t', r_t') \leftarrow$ UPDATEEDGE$(G_{t-1}, u, v, type_t, z_{t-1}, y_{t-1}, r_{t-1})$
  Update G_{t-1} to G_t using Event_t
  $(z_t, y_t, r_t) \leftarrow$ CLEANUP(G_t, z_t', y_t', r_t')
end for
return (z_k, y_k, r_k)

2.6 Time Analysis

We are now ready to analyze the running time of Algorithm 4. We begin by introducing the potential function $\Psi_W(r)$, following Zhang, Lofgren, and Goel [ZLG16] and Zheng et al. [Zhe+22]. The function is defined as

$$\Psi_W(r) := \frac{\|D^\beta r\|_1}{1 - K(1-\alpha)} = \frac{1}{1 - K(1-\alpha)} \sum_{j=1}^{n} d(j)^\beta |r_j|. \tag{4}$$

We keep the subscript W to indicate that this is the potential for the current graph. However, the potential depends on W only through the current degree matrix D.

The potential is built so that each push pays for itself. A push at node i removes the residual r_i, but it adds new residual of at most $K(1-\alpha)w_{ji}|r_i|$ at each neighbor j (Theorem 2.5). With the weights $d(j)^\beta$, these additions sum to at most $K(1-\alpha)d(i)^\beta|r_i|$, because $\sum_j d(j)^\beta w_{ji} = d(i)^\beta$ (Theorem A.8 in the Appendix). So the sum $\sum_j d(j)^\beta|r_j|$ drops by at least $(1-K(1-\alpha))d(i)^\beta|r_i|$, and the factor $1/(1-K(1-\alpha))$ in Ψ_W turns this drop into $d(i)^\beta|r_i|$. CLEANUP pushes i only when $|r_i| > (1-K(1-\alpha))\epsilon\, d(i)^{1-\beta}$, so the drop is at least $(1-K(1-\alpha))\epsilon\, d(i)$. This is proportional to the $O(d(i))$ cost of the push.

We now bound the change in potential caused by applying the PUSH procedure to a node i. The proof is given in Appendix A.

Lemma 2.6. *Let (z, y, r) satisfy Invariants 1 and 2, and let (z', y', r') be obtained by pushing node i. Then $\Psi_W(r') \le \Psi_W(r) - d(i)^\beta|r_i|$.*

Summing this drop over all pushes of CLEANUP, as explained above, gives the following bound, which does not depend on the degrees beyond the cost of each push.

Lemma 2.7. *Suppose* CLEANUP *is run on a graph G with weight matrix W, and let T_{clean} be its running time. If cleanup starts from residual r^{start} and ends at residual r^{end}, then*

$$T_{clean} \le \frac{\Psi_W(r^{start}) - \Psi_W(r^{end})}{(1-K(1-\alpha))\epsilon}.$$

The proof is given in Appendix A. With the above lemma in place, we next analyze how edge events affect the potential.

Each edge update raises the potential, and CLEANUP then pays for bringing it back down. The update changes the residual only at u and v, and the size of this change depends on the values z_u and z_v. This is where B_z enters: after CLEANUP, $|z_i| \le (B_z + \epsilon)\, d(i)^{1-\beta}$ for every i (Theorem A.12 in the Appendix). The increase in potential can be split into two terms: $\Psi_{W_t}(r'_t - r_{t-1})$ and $\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1})$. We bound the first term in Theorem 2.8 and the second term in Theorem 2.9. We then combine these bounds in Theorem 2.10. We give the proofs for these lemmas and theorem in Appendix A.

Lemma 2.8. *Consider (z'_t, y'_t, r'_t) from Algorithm 4, which is the state obtained after applying Event_t to $(z_{t-1}, y_{t-1}, r_{t-1})$. Let W_{t-1}, D_{t-1} be the weight and degree matrices before this event, and let W_t, D_t be the corresponding matrices after the event. Then*

$$\Psi_{W_t}(r'_t - r_{t-1}) \le \frac{2\,(1+2K(1-\alpha))}{1-K(1-\alpha)}(B_z + \epsilon).$$

Lemma 2.9. *Consider (z'_t, y'_t, r'_t) from Algorithm 4, which is the state obtained after applying Event_t to $(z_{t-1}, y_{t-1}, r_{t-1})$. Let W_{t-1}, D_{t-1} be the weight and degree matrices before this event, and let W_t, D_t be the corresponding matrices after the event. Then*

$$\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1}) \le 2\epsilon.$$

Theorem 2.10 (Cost of one edge update)**.** *Let T_t be the total work of the update at time t. Then*

$$T_t \le \frac{2}{1-K(1-\alpha)} + \frac{2\,(1+2K(1-\alpha))\,(B_z+\epsilon)}{(1-K(1-\alpha))^2\,\epsilon} + \frac{\Psi_{W_{t-1}}(r_{t-1}) - \Psi_{W_t}(r_t)}{(1-K(1-\alpha))\,\epsilon}.$$

We now combine the preceding results to obtain the final upper bound theorem. The proof is given in Appendix A.

Theorem 2.11 (Total cost)**.** *Consider a sequence of k updates $\mathrm{Event}_1, \ldots, \mathrm{Event}_k$, and let T_{total} denote the total running time of Algorithm 4. Then*

$$T_{\mathrm{total}} \le T_{\mathrm{init}} + \frac{2k}{1-K(1-\alpha)} + \frac{2\,(1+2K(1-\alpha))\,(B_z+\epsilon)}{(1-K(1-\alpha))^2\,\epsilon}k,$$

where $T_{\mathrm{init}} = \frac{\|D_0^\beta F(\alpha s)\|_1}{(1-K(1-\alpha))^2\epsilon}$.

Our analysis of this algorithm is asymptotically tight. The following theorem shows that the amortized time of our algorithm is at least $\Omega(1/\epsilon)$, even in the linear case. We provide the proof in Appendix A.

Theorem 2.12 (Lower bound on total cost)**.** *For any sufficiently small $\epsilon > 0$ and any k, there exists an initial graph with $n = \Theta(1/\epsilon)$ nodes, a vector s with $\|s\|_\infty \leq 1$, and a sequence of k edge insertions and deletions such that Algorithm 4 requires a total work of $\Omega(k/\epsilon)$. This result holds even in the linear case where $f_i(x) = x$ for all i.*

3 Exact Dynamic Algorithm For the Linear Case

Our main result concerns approximate maintenance for nonlinear updates, but for completeness, we also provide an exact dynamic result for the linear case. When F is the identity map, the fixed point is $z^* = [I - (1-\alpha)W]^{-1}\alpha s$. The algorithm adapts the dynamic matrix inverse framework of Sankowski [San04] to the normalized matrix $W = D^{-\beta}AD^{\beta-1}$. Unlike an entry update, one edge update changes the endpoint degrees and hence affects multiple rows and columns of W. We show how to represent these changes as low-rank updates and how to incorporate them into the lazy update and rebuild scheme of Sankowski [San04]. The details are deferred to Appendix B. We write ω for the exponent of square matrix multiplication. More generally, we write $\omega(a, b, c)$ for the exponent of the running time for multiplying an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix.

Theorem 3.1 (Exact dynamic maintenance for the linear case)**.** *Assume $f_i(x) = x$ for all i, and suppose every graph has permanent self-loops. The fixed point z^* can be maintained explicitly under (possibly weighted) edge insertions and deletions, with an initialization cost $O(n^\omega)$ and worst-case update time $O\left(n^{\omega(1,\gamma,1)-\gamma} + n^{1+\gamma}\right)$ for any parameter $\gamma \in [0, 1]$. Using current rectangular matrix multiplication bounds, this gives update time $O(n^{1.5275})$.*

Proof sketch. Let $B = I - (1-\alpha)W$. An edge update changes only two rows and two columns of W, and can be written as a rank-4 update. We then apply the lazy update and periodic rebuild scheme of Sankowski [San04]. Between rebuilds, B^{-1} is maintained implicitly using the Woodbury identity, while the explicit vector $z^* = B^{-1}\alpha s$ is updated after each edge change. After k edge updates, one update costs $O(nk + k^2)$ time. Following Sankowski [San04], after n^γ updates we rebuild the inverse using the fact that only $O(n^\gamma)$ rows and columns have changed. This gives amortized update time $O(n^{\omega(1,\gamma,1)-\gamma} + n^{1+\gamma})$. The standard deamortization used by Sankowski [San04] gives the same worst-case bound. Full details are in Appendix B. □

4 Experiments

We note that our main contribution is the theoretical analysis of the dynamic algorithm (Algorithm 4). Still, we run some experiments to evaluate the practicality of our approach. We organize the experimental contribution into two parts, corresponding to the two main benefits of the model: accuracy and running time. First, in the static setting, we show that adding non-linearity improves predictive accuracy across several benchmark datasets. Second, in the dynamic setting, we evaluate whether our algorithm preserves efficient update performance and compare its running time under non-linearity with the linear algorithm of [Zhe+22]. We publicly provide our code. [3]

Although the model and analysis are presented for scalar node values, all experiments use multi-dimensional node features. We apply the propagation independently to each feature dimension, using the same graph and parameters for every dimension. All non-linear activations reported in the experiments are uniformly bounded by a constant, and K-Lipschitz with $K \leq 1$.

4.1 Non-linearity Improves Accuracy

We evaluate our approach on several common static datasets. The datasets are explained in Appendix C. Following Zheng et al. [Zhe+22], we first propagate features and then train an MLP on the propagated representations. We use the same activation at every node and pass both the propagated vector z and the pre-activation vector $y = \alpha s + (1-\alpha)Wz$ to the classifier. Hyperparameters are selected by validation accuracy using the same search space for all activation functions. We use three

[3] `https://github.com/mahdi-jfri/dynamic-nl-public-code`

parameterized activation functions: (i) scaled tanh, $\tanh(cx)/c$ with parameter c, denoted by "Sc Tanh c"; (ii) shifted tanh, $\tanh(x - c)$ with parameter c, denoted by "Sh Tanh c"; and (iii) hard tanh, $\min(c, \max(-c, x))$ with parameter c, denoted by "H Tanh c".

We compare with GCN [KW17], GAT [Vel+17], Geom-GCN [Pei+20], APPNP [GBG18], ACM-GCN [Lua+22], and GREAD [Cho+23], all of which we run ourselves on the same splits as our model. Platonov et al. [Pla+23] show that CHAMELEON and SQUIRREL contain duplicated nodes and release filtered versions without them, so we also evaluate on these two filtered datasets. Table 1 reports the results on the original datasets, and Table 6 in the Appendix reports the results on the filtered ones. On every dataset, at least one non-linear activation is more accurate than the linear model, and the gain is sometimes large, such as 11 points on WISCONSIN. On PUBMED and on both filtered datasets, our non-linear model is more accurate than every baseline.

Table 1: Classification accuracy of models, with standard errors. The best method is underlined, and the best activation function within our model is shown in green.

Method	CORN.	TEXAS	WISC.	SQUI.	ACTOR	CHAM.	CORA	CITE.	PUBM.
GCN	42.2 ± 1.9	60.5 ± 1.6	54.4 ± 1.6	30.1 ± 0.3	28.8 ± 0.2	42.9 ± 0.8	82.6 ± 0.2	71.4 ± 0.5	79.0 ± 0.2
GAT	50.6 ± 2.0	61.2 ± 2.0	54.7 ± 1.6	30.8 ± 0.5	29.9 ± 0.4	46.0 ± 0.6	83.0 ± 0.3	72.4 ± 0.3	77.3 ± 0.1
Geom-GCN	63.1 ± 1.9	68.2 ± 1.1	65.5 ± 1.1	37.4 ± 0.3	31.8 ± 0.3	59.9 ± 0.9	66.0 ± 0.3	65.1 ± 0.3	76.6 ± 0.2
APPNP	69.1 ± 1.7	76.1 ± 1.4	80.3 ± 1.5	34.3 ± 0.5	35.5 ± 0.3	51.2 ± 0.7	83.8 ± 0.2	71.6 ± 0.2	79.7 ± 0.2
ACM-GCN	77.8 ± 1.0	85.6 ± 1.2	85.9 ± 1.1	54.1 ± 0.4	36.3 ± 0.4	67.3 ± 0.6	82.0 ± 0.2	69.9 ± 0.4	79.2 ± 0.3
GREAD	73.9 ± 0.7	85.5 ± 1.2	85.1 ± 1.3	58.2 ± 0.3	37.2 ± 0.3	70.5 ± 0.5	81.6 ± 0.3	71.0 ± 0.2	78.3 ± 0.2
Linear	66.5 ± 1.5	68.8 ± 1.5	70.8 ± 1.5	41.7 ± 0.4	33.8 ± 0.2	55.1 ± 0.5	79.8 ± 0.1	66.3 ± 0.0	80.2 ± 0.1
Sc Tanh 0.5	69.2 ± 1.0	74.7 ± 1.8	79.8 ± 1.5	41.8 ± 0.5	35.4 ± 0.3	56.1 ± 0.4	80.5 ± 0.0	70.7 ± 0.0	81.6 ± 0.0
Sc Tanh 8	70.0 ± 0.7	76.0 ± 2.0	80.4 ± 1.2	39.1 ± 0.5	35.3 ± 0.3	53.3 ± 0.8	83.3 ± 0.0	69.5 ± 0.1	82.3 ± 0.0
Sh Tanh -1.2	70.7 ± 1.1	77.4 ± 1.7	81.8 ± 0.9	36.6 ± 0.5	35.6 ± 0.2	52.3 ± 0.6	52.8 ± 0.1	46.1 ± 0.2	71.8 ± 0.0
H Tanh 2.5	67.7 ± 1.4	76.2 ± 1.8	79.2 ± 1.6	41.7 ± 0.5	35.4 ± 0.3	56.2 ± 0.4	81.6 ± 0.1	70.4 ± 0.0	82.3 ± 0.0

4.2 Dynamic Graphs

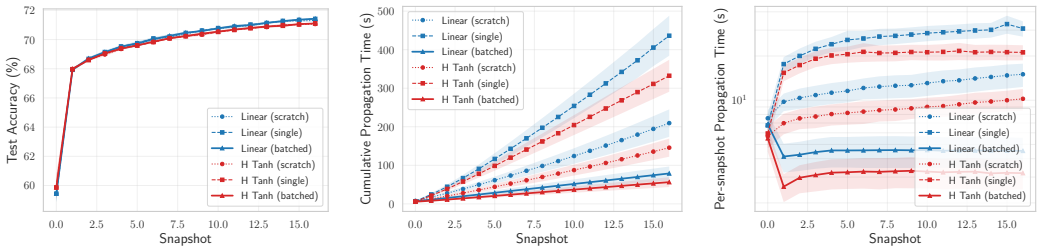


Figure 1: Results on the dynamic `arxiv` dataset. Accuracy is similar for the linear and hard-tanh models. Since this is the smallest dynamic experiment, recomputation from scratch is still faster than the "single" strategy, which handles the harder online setting of propagating after each edge update. Batched updates are much faster than both.

We evaluate dynamic propagation on `arxiv`, `products`, and `SBM-500K` [Hu+20; Zhe+22]. The first two are incremental edge-addition streams from Open Graph Benchmark, while `SBM-500K` is a synthetic dynamic graph by Zheng et al. [Zhe+22], and contains both insertions and deletions. We compare the linear model with our non-linear hard-tanh model using $c = 2.5$. Propagation is parallelized over feature dimensions using 16 CPU cores. For the linear baseline, we use the identity activation function within our framework which yields the same linear propagation model as Zheng et al. [Zhe+22]. We evaluate three update strategies: "scratch", which recomputes the propagation state after each snapshot; "single", which applies our dynamic algorithm after each individual edge update; and "batched", which processes all edge changes within a snapshot prior to cleanup. The "single" strategy aligns with the online update model analyzed in our theorems, whereas "batched" serves as a practical heuristic at the snapshot level.

Figures 1 to 3 show that the linear and non-linear models have similar propagation time and accuracy across all datasets. The dynamic methods are especially beneficial on the larger graphs, where

recomputation from scratch is much more expensive. The per-snapshot propagation time is nearly stable after the first snapshot, and cumulative propagation time grows approximately linearly with the number of snapshots, consistent with our amortized update guarantees. All curves in these experiments are averaged over 10 independent runs, and the shaded regions show 95% confidence intervals. Additional experiments in Appendix D show that the observed running times follow the predicted dependence on ϵ and α. We also run larger versions of the SBM benchmark with 2M and 5M nodes, and the `papers100M` graph from Open Graph Benchmark [Hu+20], which has 111M nodes and 1.6B edges (Table 8 in the Appendix). The advantage of dynamic updates over recomputation grows with the size of the graph. On `papers100M`, a batched update takes about two minutes per snapshot, while recomputation takes over five hours.

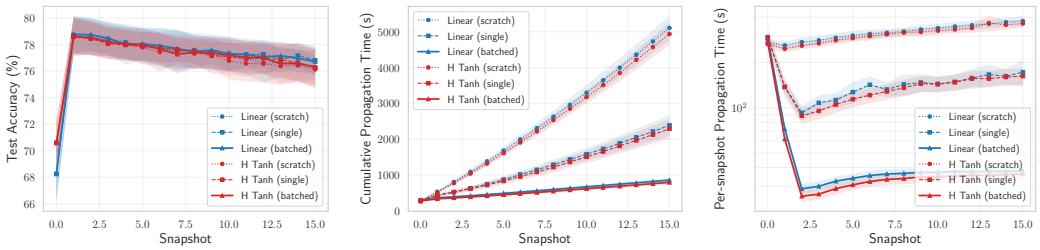


Figure 2: Results on the dynamic `products` dataset. Accuracy peaks early and then slowly decreases across later snapshots. Unlike on `arxiv`, recomputation from scratch becomes much more expensive on this larger graph, while dynamic updates keep the propagation cost substantially lower.

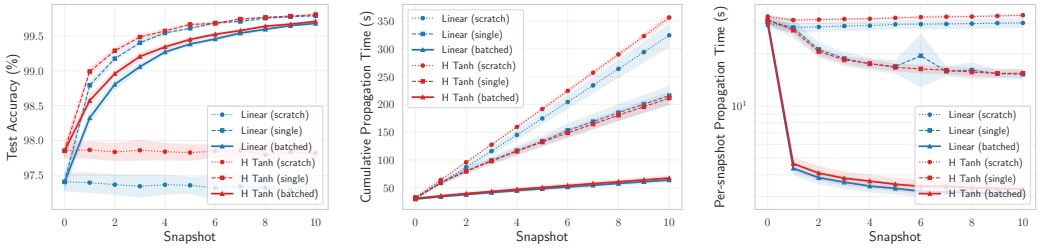


Figure 3: Results on the dynamic `SBM-500K` dataset. The dynamic methods remain efficient even with both edge insertions and deletions. After the first snapshot, their per-snapshot cost stays low and stable while preserving the accuracy behavior of the linear and hard-tanh models.

5 Conclusion

We studied dynamic maintenance of GNN representations with non-linear propagation. We gave a residual-based push algorithm with an amortized $O(1/\epsilon)$ update bound under a degree-normalized error guarantee, and we also gave an exact dynamic algorithm for the linear case. Our work has several limitations. First, the propagation is fixed and has no learned weights, so our results do not directly apply to the most popular GNNs. We leave maintaining a model with a learned matrix that mixes the features open. Second, on large dynamic graphs we compare only our linear and non-linear models. Third, we do not theoretically analyze the batched updates used in our experiments. A natural next step is to experiment more with this non-linear propagation model, including more activation functions and learned activation parameters.

Acknowledgments and Disclosure of Funding

The work is partially supported by DARPA expMath, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, NSF AF:Small grants 2218678, 2114269, 2347322. The authors declare no competing interests.

References

[ACL06] Reid Andersen, Fan Chung, and Kevin Lang. "Local graph partitioning using pagerank vectors". In: *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE. 2006, pp. 475–486.

[Alm+25] Josh Alman et al. "More asymmetry yields faster matrix multiplication". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2025, pp. 2005–2039.

[Bah+12] Bahman Bahmani et al. "Pagerank on an evolving graph". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, pp. 24–32.

[Ban22] Stefan Banach. "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales". In: *Fundamenta mathematicae* 3.1 (1922), pp. 133–181.

[BCG10] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. "Fast Incremental and Personalized PageRank". In: *Proc. VLDB Endow.* 4.3 (2010), pp. 173–184. DOI: 10.14778/1929861.1929864. URL: http://www.vldb.org/pvldb/vol4/p173-bahmani.pdf.

[BH74] James R Bunch and John E Hopcroft. "Triangular factorization and inversion by fast matrix multiplication". In: *Mathematics of Computation* 28.125 (1974), pp. 231–236.

[Cho+23] Jeongwhan Choi et al. "GREAD: Graph Neural Reaction-Diffusion Networks". In: *Proceedings of the 40th International Conference on Machine Learning*. Vol. 202. Proceedings of Machine Learning Research. 2023.

[Cra+98] Mark Craven et al. *Learning to extract symbolic knowledge from the world wide web*. Tech. rep. 1998.

[FL19] Matthias Fey and Jan Eric Lenssen. "Fast graph representation learning with PyTorch Geometric". In: *arXiv preprint arXiv:1903.02428* (2019).

[GBG18] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. "Predict then propagate: Graph neural networks meet personalized pagerank". In: *arXiv preprint arXiv:1810.05997* (2018).

[Hu+20] Weihua Hu et al. "Open graph benchmark: Datasets for machine learning on graphs". In: *Advances in neural information processing systems* 33 (2020), pp. 22118–22133.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[Jay+24] Rajesh Jayaram et al. "Dynamic PageRank: Algorithms and Lower Bounds". In: *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, Tallinn, Estonia, July 8-12, 2024*. Ed. by Karl Bringmann et al. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 90:1–90:19. DOI: 10.4230/LIPICS.ICALP.2024.90. URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.90.

[KW17] Thomas N Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations*. 2017.

[KZL18] Srijan Kumar, Xikun Zhang, and Jure Leskovec. "Learning dynamic embeddings from temporal interactions". In: *arXiv preprint arXiv:1812.02289* (2018).

[Lua+22] Sitao Luan et al. "Revisiting heterophily for graph neural networks". In: *Advances in neural information processing systems* 35 (2022), pp. 1362–1375.

[Nam+12] Galileo Namata et al. "Query-driven active surveying for collective classification". In: *10th international workshop on mining and learning with graphs*. Vol. 8. 2012, p. 1.

[Pei+20] Hongbin Pei et al. "Geom-GCN: Geometric Graph Convolutional Networks". In: *International Conference on Learning Representations*. 2020.

[Pla+23] Oleg Platonov et al. "A critical look at the evaluation of GNNs under heterophily: Are we really making progress?" In: *arXiv preprint arXiv:2302.11640* (2023).

[RAS21] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. "Multi-scale attributed node embedding". In: *Journal of Complex Networks* 9.2 (2021), cnab014.

[Ros+20] Emanuele Rossi et al. "Temporal graph networks for deep learning on dynamic graphs". In: *arXiv preprint arXiv:2006.10637* (2020).

[San04] Piotr Sankowski. "Dynamic transitive closure via dynamic matrix inverse". In: *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE. 2004, pp. 509–517.

[Sca+08] Franco Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[Sen+08] Prithviraj Sen et al. "Collective classification in network data". In: *AI magazine* 29.3 (2008), pp. 93–93.

[Tan+09] Jie Tang et al. "Social influence analysis in large-scale networks". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2009, pp. 807–816.

[Tri+19] Rakshit Trivedi et al. "Dyrep: Learning representations over dynamic graphs". In: *International conference on learning representations*. 2019.

[Vel+17] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[VNS19] Jan Van Den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. "Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds". In: *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2019, pp. 456–480.

[Woo50] Max A Woodbury. *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

[Wu+20] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[Xu+20] da Xu et al. "Inductive representation learning on temporal graphs". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rJeW1yHYwH.

[YCS16] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. "Revisiting semi-supervised learning with graph embeddings". In: *International conference on machine learning*. PMLR. 2016, pp. 40–48.

[Zhe+22] Yanping Zheng et al. "Instant graph neural networks for dynamic graphs". In: *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*. 2022, pp. 2605–2615.

[ZLG16] Hongyang Zhang, Peter Lofgren, and Ashish Goel. "Approximate personalized pagerank on dynamic graphs". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1315–1324.

[ZWL23] Yanping Zheng, Zhewei Wei, and Jiajun Liu. "Decoupled graph neural networks for large dynamic graphs". In: *arXiv preprint arXiv:2305.08273* (2023).

[ZYW25] Yanping Zheng, Lu Yi, and Zhewei Wei. "A survey of dynamic graph neural networks". In: *Frontiers of Computer Science* 19.6 (2025), p. 196323.

A Omitted Lemmas and Proofs of Section 2

We restate the lemmas omitted from the main text and provide their proofs. We also introduce several auxiliary lemmas used in later proofs. The results are ordered by their logical dependency and their sequence in the main text.

Table 2 lists the parameters used in the paper and their roles.

Lemma A.1. *If $f_i(x) \in [-B_z, B_z]$ for all i and x, then $\|D^{\beta-1}z^*\|_\infty \le B_z$.*

Proof. Since $f_i(x) \in [-B_z, B_z]$ for all i and x, each coordinate of $F(x)$ also lies in $[-B_z, B_z]$. Because $z^* = F(\alpha s + (1-\alpha)Wz^*)$, it follows that $z_i^* \in [-B_z, B_z]$ for every i. Since $d(i) \ge 1$ and $\beta \in [0,1]$, we have $d(i)^{\beta-1} \le 1$. Hence

$$\|D^{\beta-1}z^*\|_\infty = \max_i d(i)^{\beta-1}|z_i^*| \le \max_i |z_i^*| \le B_z. \qquad \square$$

Lemma A.2. *For any vector v, $\|Wv\|_{D,\infty} \le \|v\|_{D,\infty}$.*

Table 2: Parameters used in the paper.

Symbol	Meaning
n	Number of nodes.
A	Adjacency matrix of the current graph, with a self-loop at every node.
D	Diagonal degree matrix of the current graph.
$d(i)$	Degree of node i. Since every node has a self-loop, $d(i) \geq 1$.
$\beta \in [0, 1]$	Normalization parameter in $W = D^{-\beta} A D^{\beta-1}$.
W	Normalized weight matrix.
$\alpha \in (0, 1)$	Teleportation probability, the weight a node keeps on its own input at each step.
s	Input values.
f_i, F	Activation at node i. F applies f_i to coordinate i.
$K \leq 1$	Every f_i is K-Lipschitz.
T, z^*	The map $T(z) = F(\alpha s + (1-\alpha)Wz)$ and its unique fixed point.
$\|\cdot\|_{D,\infty}$	Degree-scaled infinity norm, $\|v\|_{D,\infty} = \|D^{\beta-1} v\|_\infty$.
B_z	Bound on $\|z^*\|_{D,\infty}$ on every graph.
ϵ	Accuracy. The algorithm keeps $\|z_i - z_i^*\| \leq \epsilon\, d(i)^{1-\beta}$ for every i.
z, y, r	Maintained state: the estimate of z^*, the pre-activation vector $\alpha s + (1-\alpha)Wz$, and the residual $F(y) - z$.
Ψ_W	Potential $\sum_j d(j)^\beta \|r_j\| / (1 - K(1-\alpha))$, used to bound the running time.
k	Number of edge updates.
ω, $\omega(a, b, c)$, γ	Matrix multiplication exponents and the rebuild parameter of the exact algorithm in Appendix B.

Proof. Since $W = D^{-\beta} A D^{\beta-1}$,

$$\begin{aligned} \|Wv\|_{D,\infty} &= \|D^{\beta-1} D^{-\beta} A D^{\beta-1} v\|_\infty \\ &= \|D^{-1} A D^{\beta-1} v\|_\infty \\ &\leq \|D^{-1} A\|_\infty \|v\|_{D,\infty}. \end{aligned}$$

Moreover, $\|D^{-1}A\|_\infty = \max_i d(i)^{-1} \sum_j A_{ij} = 1$. Thus $\|Wv\|_{D,\infty} \leq \|v\|_{D,\infty}$. □

The next lemma bounds B_z for activations that may be unbounded, such as ReLU.

Lemma A.3. *The fixed point z^* satisfies*

$$\|D^{\beta-1} z^*\|_\infty \leq \frac{\max_i |f_i(0)| + K\alpha \|s\|_\infty}{1 - K(1-\alpha)}.$$

Proof. Fix a node i, and let $x_i = \alpha s_i + (1-\alpha)(Wz^*)_i$, so that $z_i^* = f_i(x_i)$. Since f_i is K-Lipschitz,

$$|z_i^*| \leq |f_i(0)| + K|x_i| \leq \max_j |f_j(0)| + K\alpha |s_i| + K(1-\alpha)|(Wz^*)_i|.$$

We multiply both sides by $d(i)^{\beta-1}$, which is at most 1 because $d(i) \geq 1$ and $\beta \leq 1$. The first term stays at most $\max_j |f_j(0)|$. The second term becomes at most $K\alpha\|s\|_\infty$. The third term becomes at most $K(1-\alpha)\|Wz^*\|_{D,\infty}$, which is at most $K(1-\alpha)\|z^*\|_{D,\infty}$ by Theorem 2.1. Taking the maximum over i gives

$$\|z^*\|_{D,\infty} \leq \max_j |f_j(0)| + K\alpha\|s\|_\infty + K(1-\alpha)\|z^*\|_{D,\infty}.$$

Since $K(1-\alpha) < 1$, rearranging gives the claim. □

Lemma A.4. *Assume that $\alpha \in (0, 1)$, $\beta \in [0, 1]$. Further assume that each f_i is K-Lipschitz with $K \leq 1$. Then there is a unique fixed point z^* satisfying $T(z^*) = z^*$. Moreover, the iteration $z(t+1) = T(z(t))$ converges to z^* for any initial point $z(0)$ as $t \to \infty$.*

Proof. For any $x, y \in \mathbb{R}^n$,

$$\|T(x) - T(y)\|_{D,\infty} = \|F(\alpha s + (1-\alpha)Wx) - F(\alpha s + (1-\alpha)Wy)\|_{D,\infty}.$$

Since each f_i is K-Lipschitz, the pointwise map F is also K-Lipschitz under $\|\cdot\|_{D,\infty}$. Therefore,

$$\|T(x) - T(y)\|_{D,\infty} \le K(1-\alpha)\|W(x-y)\|_{D,\infty}.$$

By Theorem 2.1, $\|T(x) - T(y)\|_{D,\infty} \le K(1-\alpha)\|x-y\|_{D,\infty}$. Since $\alpha \in (0,1)$ and $K \le 1$, we have $K(1-\alpha) < 1$. Hence T is a contraction under the degree-scaled infinity norm. The Banach fixed-point theorem [Ban22] implies that the fixed point is unique and that the iteration $z(t+1) = T(z(t))$ converges to it from any initial point. □

Lemma A.5. *Let (z, y, r) satisfy Invariants 1 and 2. Then, for every i, $|z_i - z_i^*| \le \frac{d(i)^{1-\beta}}{1-K(1-\alpha)}\|D^{\beta-1}r\|_\infty$.*

Proof. For any $x, y \in \mathbb{R}^n$,

$$\begin{aligned}\|F(x) - F(y)\|_{D,\infty} &= \max_i \frac{|f_i(x_i) - f_i(y_i)|}{d(i)^{1-\beta}} \\ &\le K\|x-y\|_{D,\infty},\end{aligned}$$

because each f_i is K-Lipschitz. Together with Theorem 2.1, this gives $\|T(x) - T(y)\|_{D,\infty} \le K(1-\alpha)\|x-y\|_{D,\infty}$. By Invariants 1 and 2, we have $z + r = F(y) = T(z)$. Subtracting this identity from $z^* = T(z^*)$ gives $z - z^* = T(z) - T(z^*) - r$. Thus

$$\begin{aligned}\|z - z^*\|_{D,\infty} &\le \|T(z) - T(z^*)\|_{D,\infty} + \|r\|_{D,\infty} \\ &\le K(1-\alpha)\|z - z^*\|_{D,\infty} + \|r\|_{D,\infty}.\end{aligned}$$

Rearranging, $\|z - z^*\|_{D,\infty} \le \frac{\|r\|_{D,\infty}}{1-K(1-\alpha)}$. Therefore, for each i,

$$|z_i - z_i^*| \le \frac{d(i)^{1-\beta}}{1-K(1-\alpha)}\|D^{\beta-1}r\|_\infty.$$ □

Lemma A.6. *Suppose* PUSH(i, G, z, y, r) *is called on a state (z, y, r) that satisfies Invariants 1 and 2, and let (z', y', r') be the new state. Then (z', y', r') also satisfies Invariants 1 and 2. Moreover, for all $j \ne i$, $|r'_j| \le |r_j| + K(1-\alpha)w_{ji}|r_i|$. Furthermore, $|r'_i| \le K(1-\alpha)w_{ii}|r_i|$.*

Proof. Only coordinate i of z changes, and $z'_i - z_i = r_i$. Hence, for every j,

$$\begin{aligned}y'_j &= y_j + (1-\alpha)w_{ji}r_i \\ &= \alpha s_j + (1-\alpha)\sum_k w_{jk}z'_k.\end{aligned}$$

Thus Invariant 1 is preserved. By the definition of r', $z'_j + r'_j = f_j(y'_j)$ for every j, so Invariant 2 is also preserved.

For $j \ne i$,

$$|r'_j| = |f_j(y'_j) - z_j| \le |f_j(y'_j) - f_j(y_j)| + |f_j(y_j) - z_j| = K(1-\alpha)w_{ji}|r_i| + |r_j|.$$

For i, using $z_i + r_i = f_i(y_i)$,

$$|r'_i| = |f_i(y'_i) - z_i - r_i| = |f_i(y'_i) - f_i(y_i)| \le K|y'_i - y_i| = K(1-\alpha)w_{ii}|r_i|.$$

This proves the claim. □

Lemma A.7. *Suppose* UPDATEEDGE$(G, u, v, type, z, y, r)$ *is called on a state (z, y, r) satisfying Invariants 1 and 2, and let (z', y', r') be the returned state. Then (z', y', r') satisfies Invariants 1 and 2.*

Proof. Let $d'_u = d_u + \sigma$ and $d'_v = d_v + \sigma$ be the new degrees. By Invariant 1, before the update, $S_u = \frac{d_u^\beta}{1-\alpha}(y_u - \alpha s_u) = \sum_{j \in N(u)} \frac{z_j}{d(j)^{1-\beta}}$, and the same identity holds for S_v.

The rescaling step preserves the degree-scaled values of the endpoints:

$$\frac{z'_u}{(d'_u)^{1-\beta}} = \frac{z_u}{d_u^{1-\beta}} = x_u, \qquad \frac{z'_v}{(d'_v)^{1-\beta}} = \frac{z_v}{d_v^{1-\beta}} = x_v.$$

Thus the contributions of u and v to each non-endpoint neighbor do not change. Hence, for every $w \notin \{u, v\}$, Invariant 1 continues to hold with $y'_w = y_w$. Since $z'_w = z_w$ and $r'_w = r_w$, Invariant 2 also continues to hold at such nodes.

It remains to check the endpoints. After the update, the scaled neighbor sum at u is $S_u + \sigma x_v$. Therefore $y'_u = \alpha s_u + (1-\alpha)(S_u + \sigma x_v)/(d_u + \sigma)^\beta$ is exactly the value required by Invariant 1 at u. The same argument applies to v. Finally, setting $r'_u = f_u(y'_u) - z'_u$ and $r'_v = f_v(y'_v) - z'_v$ gives Invariant 2 at the two endpoints. Therefore the returned state satisfies both invariants. □

Lemma A.8. *For every node i, $\sum_j d(j)^\beta w_{ji} = d(i)^\beta$.*

Proof. By the definition of W, $d(j)^\beta w_{ji} = d(j)^\beta \cdot \frac{A_{ji}}{d(j)^\beta d(i)^{1-\beta}} = \frac{A_{ji}}{d(i)^{1-\beta}}$. Summing over j gives $\sum_j d(j)^\beta w_{ji} = \frac{\sum_j A_{ji}}{d(i)^{1-\beta}} = \frac{d(i)}{d(i)^{1-\beta}} = d(i)^\beta$. □

Lemma A.9. *Let (z, y, r) satisfy Invariants 1 and 2, and let (z', y', r') be obtained by pushing node i. Then $\Psi_W(r') \le \Psi_W(r) - d(i)^\beta |r_i|$.*

Proof. First, note that $(1 - K(1-\alpha))\,\Psi_W(r') = \sum_{j=1}^n d(j)^\beta |r'_j|$. Applying Theorem 2.5 to r' gives $\sum_{j=1}^n d(j)^\beta |r'_j| \le \sum_{j \ne i} d(j)^\beta \left(|r_j| + K(1-\alpha) w_{ji} |r_i|\right) + d(i)^\beta K(1-\alpha) w_{ii} |r_i|$. The right-hand side is equal to $\sum_{j \ne i} d(j)^\beta |r_j| + K(1-\alpha)|r_i| \sum_{j=1}^n d(j)^\beta w_{ji}$. By Theorem A.8, and after adding and subtracting $d(i)^\beta |r_i|$, we obtain

$$\begin{aligned}(1 - K(1-\alpha))\,\Psi_W(r') &\le (1 - K(1-\alpha))\,\Psi_W(r) - d(i)^\beta |r_i| + K(1-\alpha) d(i)^\beta |r_i| \\ &= (1 - K(1-\alpha))\,\Psi_W(r) - (1 - K(1-\alpha))\, d(i)^\beta |r_i|.\end{aligned}$$

Dividing by $1 - K(1-\alpha)$ gives the claim. □

Lemma A.10. *Suppose* CLEANUP *is run on a graph G with weight matrix W, and let T_{clean} be its running time. If cleanup starts from residual r^{start} and ends at residual r^{end}, then*

$$T_{clean} \le \frac{\Psi_W(r^{start}) - \Psi_W(r^{end})}{(1 - K(1-\alpha))\epsilon}.$$

Proof. First note that $T_{\text{clean}} = \sum_{\text{pushes at } i} d(i)$. By Theorem 2.6, a push at node i decreases Ψ_W by at least $d(i)^\beta |r_i|$. Since CLEANUP only pushes when $|r_i| > (1 - K(1-\alpha))\epsilon d(i)^{1-\beta}$, the decrease from this push is at least $d(i)^\beta |r_i| > (1 - K(1-\alpha))\epsilon d(i)$. Summing this bound over all pushes gives the result. □

Lemma A.11. *Let $d > 0$, let $\delta \in \mathbb{R}$ satisfy $d + \delta > 0$, and let $\beta \in [0, 1]$. Then $d^{1-\beta}|(d+\delta)^\beta - d^\beta| \le |\delta|$. Equivalently, $|(d+\delta)^\beta - d^\beta| \le \frac{|\delta|}{d^{1-\beta}}$. In particular, when $\delta = 1$, $d^{1-\beta}\left((d+1)^\beta - d^\beta\right) \le 1$.*

Proof. Let $x := \delta/d$. Then $x > -1$, and

$$d^{1-\beta}|(d+\delta)^\beta - d^\beta| = d^{1-\beta} d^\beta |(1+x)^\beta - 1| = d|(1+x)^\beta - 1|.$$

It is enough to show that $|(1+x)^\beta - 1| \le |x|$ for all $x > -1$.

If $x \ge 0$, Bernoulli's inequality gives $(1+x)^\beta \le 1 + x$. Therefore, $0 \le (1+x)^\beta - 1 \le x = |x|$. If $-1 < x \le 0$, set $y := 1 + x \in (0, 1]$. Since $\beta \in [0, 1]$, we have $y^\beta \ge y$. Hence $0 \le 1 - y^\beta \le 1 - y = -x = |x|$. Equivalently, $|(1+x)^\beta - 1| \le |x|$. Thus $d|(1+x)^\beta - 1| \le d|x| = |\delta|$, which proves the claim. □

Lemma A.12. *Suppose z satisfies $\|z - z^*\|_{D,\infty} \le \epsilon$. Then $\|z\|_{D,\infty} \le B_z + \epsilon$.*

Proof. By assumption, $\|z^*\|_{D,\infty} \le B_z$. Therefore, $\|z\|_{D,\infty} \le \|z^*\|_{D,\infty} + \|z - z^*\|_{D,\infty} \le B_z + \epsilon$. □

Lemma A.13. *Consider (z'_t, y'_t, r'_t) from Algorithm 4, which is the state obtained after applying* Event_t *to $(z_{t-1}, y_{t-1}, r_{t-1})$. Let W_{t-1}, D_{t-1} be the weight and degree matrices before this event, and let W_t, D_t be the corresponding matrices after the event. Then*

$$\Psi_{W_t}(r'_t - r_{t-1}) \le \frac{2\,(1 + 2K(1-\alpha))}{1 - K(1-\alpha)}(B_z + \epsilon).$$

Proof. Let $\sigma \in \{+1, -1\}$, where $\sigma = +1$ denotes an insertion and $\sigma = -1$ denotes a deletion. Write $a = d_{t-1}(u)$ and $b = d_{t-1}(v)$. Then $d_t(u) = a + \sigma$ and $d_t(v) = b + \sigma$. Since self-loops are permanent and only the non-self edge (u, v) is updated, both $a + \sigma$ and $b + \sigma$ are positive. Define $x_i := d_{t-1}(i)^{\beta-1} z_{t-1,i}$ and $X := \|z_{t-1}\|_{D_{t-1},\infty} = \|x\|_\infty$. Since CLEANUP has been applied before event t, Theorem A.12 gives $X \le B_z + \epsilon$. By Algorithm 3, the vector $r'_t - r_{t-1}$ is supported only on u and v. In the procedure, $S_v = \sum_{j \in N_{t-1}(v)} x_j$ and $S_u = \sum_{j \in N_{t-1}(u)} x_j$. Thus $|S_v| \le bX$ and $|S_u| \le aX$. We prove the bound for v; the proof for u is the same.

Before the update, $y_{t-1,v} = \alpha s_v + (1-\alpha) b^{-\beta} S_v$. After the update, $y'_v = \alpha s_v + (1-\alpha)(b + \sigma)^{-\beta}(S_v + \sigma x_u)$. Therefore,

$$y'_v - y_{t-1,v} = (1-\alpha)\left[\left(\frac{1}{(b+\sigma)^\beta} - \frac{1}{b^\beta}\right) S_v + \frac{\sigma x_u}{(b+\sigma)^\beta}\right].$$

By Invariant 2, $r'_{t,v} - r_{t-1,v} = (f_v(y'_v) - z'_v) - (f_v(y_{t-1,v}) - z_{t-1,v})$. Using the K-Lipschitz property of f_v, $|r'_{t,v} - r_{t-1,v}| \le |z'_v - z_{t-1,v}| + K|y'_v - y_{t-1,v}|$. Multiplying by $(b+\sigma)^\beta$ gives

$$\begin{aligned}(b+\sigma)^\beta |r'_{t,v} - r_{t-1,v}| \le{}& (b+\sigma)^\beta |z'_v - z_{t-1,v}| \\ &+ K(1-\alpha)(b+\sigma)^\beta \left|\frac{1}{(b+\sigma)^\beta} - \frac{1}{b^\beta}\right| |S_v| \\ &+ K(1-\alpha)|x_u|.\end{aligned}$$

We bound the three terms on the right. Since $z'_v = (b+\sigma)^{1-\beta} x_v$ and $z_{t-1,v} = b^{1-\beta} x_v$, the first term is

$$(b+\sigma)^\beta \left|(b+\sigma)^{1-\beta} - b^{1-\beta}\right| |x_v|.$$

Applying Theorem A.11 with $d = b + \sigma$, $\delta = -\sigma$, and exponent $1 - \beta$ gives $(b+\sigma)^\beta |(b+\sigma)^{1-\beta} - b^{1-\beta}| \le 1$. Thus the first term is at most X. For the second term,

$$(b+\sigma)^\beta \left|\frac{1}{(b+\sigma)^\beta} - \frac{1}{b^\beta}\right| |S_v| \le \frac{|(b+\sigma)^\beta - b^\beta|}{b^\beta} bX = b^{1-\beta} |(b+\sigma)^\beta - b^\beta| X.$$

Applying Theorem A.11 with $d = b$, $\delta = \sigma$, and exponent β gives $b^{1-\beta}|(b+\sigma)^\beta - b^\beta| \le 1$. Thus the second term is at most $K(1-\alpha)X$. The third term is also at most $K(1-\alpha)X$, since $|x_u| \le X$. Combining these bounds,

$$d_t(v)^\beta |r'_{t,v} - r_{t-1,v}| = (b+\sigma)^\beta |r'_{t,v} - r_{t-1,v}| \le (1 + 2K(1-\alpha))\, X.$$

The same bound holds for u. Hence

$$\begin{aligned}\Psi_{W_t}(r'_t - r_{t-1}) &= \frac{d_t(u)^\beta |r'_{t,u} - r_{t-1,u}| + d_t(v)^\beta |r'_{t,v} - r_{t-1,v}|}{1 - K(1-\alpha)} \\ &\le \frac{2\,(1 + 2K(1-\alpha))}{1 - K(1-\alpha)} X.\end{aligned}$$

Using $X \le B_z + \epsilon$ gives the claim. □

Lemma A.14. *Consider (z'_t, y'_t, r'_t) from Algorithm 4, which is the state obtained after applying* Event_t *to $(z_{t-1}, y_{t-1}, r_{t-1})$. Let W_{t-1}, D_{t-1} be the weight and degree matrices before this event, and let W_t, D_t be the corresponding matrices after the event. Then*

$$\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1}) \leq 2\epsilon.$$

Proof. By the definition of Ψ_W,

$$\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1}) = \frac{1}{1-K(1-\alpha)} \sum_i \left(d_t(i)^\beta - d_{t-1}(i)^\beta\right) |r_{t-1}(i)|.$$

Only the endpoints u and v can change degree. If the event is a deletion, then $d_t(i) \leq d_{t-1}(i)$ for $i \in \{u, v\}$, so the above quantity is non-positive. It remains to consider an insertion. For $i \in \{u, v\}$, we have $d_t(i) = d_{t-1}(i)+1$. Since cleanup has terminated before the update, $|r_{t-1}(i)| \leq (1-K(1-\alpha))\,\epsilon d_{t-1}(i)^{1-\beta}$. Therefore,

$$\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1}) \leq \epsilon \sum_{i\in\{u,v\}} d_{t-1}(i)^{1-\beta} \left((d_{t-1}(i)+1)^\beta - d_{t-1}(i)^\beta\right).$$

By Theorem A.11, applied with $d = d_{t-1}(i)$ and $\delta = 1$,

$$d_{t-1}(i)^{1-\beta} \left((d_{t-1}(i)+1)^\beta - d_{t-1}(i)^\beta\right) \leq 1.$$

Thus, in all cases, $\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1}) \leq 2\epsilon$. □

Theorem 2.10 (Cost of one edge update)**.** *Let T_t be the total work of the update at time t. Then*

$$T_t \leq \frac{2}{1-K(1-\alpha)} + \frac{2\left(1+2K(1-\alpha)\right)\left(B_z+\epsilon\right)}{\left(1-K(1-\alpha)\right)^2\epsilon} + \frac{\Psi_{W_{t-1}}(r_{t-1}) - \Psi_{W_t}(r_t)}{\left(1-K(1-\alpha)\right)\epsilon}.$$

Proof. By Theorem 2.7 applied to the new graph G_t,

$$T_t \leq \frac{\Psi_{W_t}(r'_t) - \Psi_{W_t}(r_t)}{\left(1-K(1-\alpha)\right)\epsilon}.$$

The triangle inequality gives $\Psi_{W_t}(r'_t) \leq \Psi_{W_t}(r_{t-1}) + \Psi_{W_t}(r'_t - r_{t-1})$.

Combining this with $\Psi_{W_t}(r_{t-1}) = \Psi_{W_{t-1}}(r_{t-1}) + \left(\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1})\right)$, we obtain

$$T_t \leq \frac{\Psi_{W_{t-1}}(r_{t-1}) - \Psi_{W_t}(r_t)}{\left(1-K(1-\alpha)\right)\epsilon} + \frac{\Psi_{W_t}(r'_t - r_{t-1})}{\left(1-K(1-\alpha)\right)\epsilon} + \frac{\Psi_{W_t}(r_{t-1}) - \Psi_{W_{t-1}}(r_{t-1})}{\left(1-K(1-\alpha)\right)\epsilon}.$$

The first term is kept in the final bound. The second term is bounded by Theorem 2.8, and the third term is bounded by Theorem 2.9. Combining these bounds gives the claim. □

Theorem 2.11 (Total cost)**.** *Consider a sequence of k updates* $\mathrm{Event}_1, \ldots, \mathrm{Event}_k$*, and let T_{total} denote the total running time of Algorithm 4. Then*

$$T_{\mathrm{total}} \leq T_{\mathrm{init}} + \frac{2k}{1-K(1-\alpha)} + \frac{2\left(1+2K(1-\alpha)\right)\left(B_z+\epsilon\right)}{\left(1-K(1-\alpha)\right)^2\epsilon} k,$$

where $T_{\mathrm{init}} = \frac{\|D_0^\beta F(\alpha s)\|_1}{(1-K(1-\alpha))^2\epsilon}$.

Proof. The total running time is the sum of the initialization cost and the cost of all edge updates. By Theorem 2.7, the initialization cost is at most

$$\frac{\Psi_{W_0}(r_0^{\mathrm{start}}) - \Psi_{W_0}(r_0)}{\left(1-K(1-\alpha)\right)\epsilon}.$$

For the update costs, summing the bound of Theorem 2.10 over $t = 1, \ldots, k$ gives

$$\sum_{t=1}^k T_t \leq \frac{2k}{1-K(1-\alpha)} + \frac{2\left(1+2K(1-\alpha)\right)}{\left(1-K(1-\alpha)\right)^2\epsilon} \sum_{t=1}^k \|D_{t-1}^{\beta-1} z_{t-1}\|_\infty + \frac{1}{\left(1-K(1-\alpha)\right)\epsilon} \sum_{t=1}^k \left(\Psi_{W_{t-1}}(r_{t-1}) - \Psi_{W_t}(r_t)\right).$$

The potential terms telescope:

$$\sum_{t=1}^{k} \left(\Psi_{W_{t-1}}(r_{t-1}) - \Psi_{W_t}(r_t)\right) = \Psi_{W_0}(r_0) - \Psi_{W_k}(r_k) \leq \Psi_{W_0}(r_0).$$

Therefore, including initialization,

$$T_{\text{total}} \leq \frac{\Psi_{W_0}(r_0^{\text{start}}) - \Psi_{W_0}(r_0)}{(1 - K(1-\alpha))\,\epsilon} + \frac{\Psi_{W_0}(r_0)}{(1 - K(1-\alpha))\,\epsilon} + \frac{2k}{1 - K(1-\alpha)} + \frac{2\,(1 + 2K(1-\alpha))}{(1 - K(1-\alpha))^2\,\epsilon} \sum_{t=1}^{k} \|D_{t-1}^{\beta-1} z_{t-1}\|_\infty.$$

Also, $\Psi_{W_0}(r_0^{\text{start}}) = \frac{\|D_0^\beta F(\alpha s)\|_1}{1-K(1-\alpha)}$. By Theorem A.12, $\|D_{t-1}^{\beta-1} z_{t-1}\|_\infty \leq B_z + \epsilon$ for every t. Substituting this bound gives

$$T_{\text{total}} \leq T_{\text{init}} + \frac{2k}{1 - K(1-\alpha)} + \frac{2\,(1 + 2K(1-\alpha))\,(B_z + \epsilon)}{(1 - K(1-\alpha))^2\,\epsilon} k. \qquad \square$$

Theorem 2.12 (Lower bound on total cost)**.** *For any sufficiently small $\epsilon > 0$ and any k, there exists an initial graph with $n = \Theta(1/\epsilon)$ nodes, a vector s with $\|s\|_\infty \leq 1$, and a sequence of k edge insertions and deletions such that Algorithm 4 requires a total work of $\Omega(k/\epsilon)$. This result holds even in the linear case where $f_i(x) = x$ for all i.*

Proof. We give a construction in which the same edge is inserted and deleted repeatedly. Each insertion forces a push at a node of degree $\Theta(1/\epsilon)$.

Set $f_i(x) = x$, $\alpha = 1/2$, and $\beta = 0$. Then $K = 1$, and the cleanup threshold at node i is $|r_i| > \frac{1}{2}\epsilon d(i)$. We have $W = AD^{-1}$, so a node j contributes $z_j/d(j)$ to each of its neighbors.

Our graph consists of a star and an isolated vertex. Let u be the center of the star, and let its degree be

$$d(u) = m := \left\lfloor \frac{1}{100\epsilon} \right\rfloor$$

including its permanent self-loop. Thus u has $m - 1$ leaves, and each leaf has degree 2. Let v be the isolated vertex except for its permanent self-loop, which is initially disconnected from the star. Hence the graph has $n = m + 1 = \Theta(1/\epsilon)$ nodes.

Define the s vector by $s_v = 1$ and $s_i = 0$ for all $i \neq v$. In this initial graph, the fixed point has $z_v^* = 1$ and $z_i^* = 0$ for all $i \neq v$. After cleanup, the maintained state therefore satisfies $|z_u| \leq \epsilon m$ and $|z_v - 1| \leq \epsilon$ and all residuals are below threshold. In particular, $|r_u| \leq \frac{1}{2}\epsilon m$ and $|r_v| \leq \frac{1}{2}\epsilon$.

Now insert the edge (u, v). We show that the subsequent cleanup must push u. The update rule scales z_u by $(m+1)/m$ and adds the new contribution from v to u. Hence the new residual at u is

$$r_u' = r_u + \frac{1}{2} z_v - \frac{z_u}{m}.$$

Using the bounds above and $\epsilon m \leq 1/100$, we get for $\epsilon \leq 1/100$,

$$\begin{aligned} r_u' &\geq -\frac{1}{2}\epsilon m + \frac{1}{2}(1 - \epsilon) - \epsilon \\ &\geq -\frac{1}{200} + \frac{1}{2}\left(1 - \frac{1}{100}\right) - \frac{1}{100} \\ &= 0.48. \end{aligned}$$

On the other hand, the cleanup threshold at u after the insertion is $\frac{1}{2}\epsilon(m+1) \leq 0.01$, thus u has a large residual immediately after the edge update. It remains to check that cleanup cannot remove this residual by pushing other nodes first. Before u is pushed, the pushes that can affect r_u are pushes at its neighbors.

First consider node v. After the insertion, $d(v) = 2$, so pushing v with residual r'_v changes r_u by $(1-\alpha)w_{uv}r'_v = r'_v/4$. The residual left at v after this push is also $r'_v/4$. Therefore, even if v is pushed repeatedly before u is ever pushed, its total possible effect on r_u has magnitude at most

$$\frac{1}{4}|r'_v| + \frac{1}{16}|r'_v| + \frac{1}{64}|r'_v| + \cdots = \frac{1}{3}|r'_v|.$$

After the insertion, $r'_v = r_v - z_v + \frac{1}{2} \cdot \frac{z_u}{m}$. Using bounds before the insertion we have $|r'_v| \leq \epsilon/2 + (1+\epsilon) + \epsilon/2 = 1 + 2\epsilon$, so all pushes of v can reduce r_u by at most $(1+2\epsilon)/3 \leq 0.34$.

Now consider the previous leaves. The insertion of (u, v) does not change their residuals, because the scaling step preserves the quantity $z_u/d(u)$ that they receive from u. Thus each leaf i still has $|r_i| \leq \epsilon$, since $d(i) = 2$. The same argument as v shows that all pushes of one leaf can affect r_u by at most $|r_i|/3$. Summing over the $m-1$ leaves, the total possible effect from all leaves is at most $\frac{1}{3}(m-1)\epsilon \leq \frac{1}{300}$.

Therefore, before the first push of u, even after all possible pushes of v and all leaves, the residual at u is still at least $0.48 - 0.34 - \frac{1}{300} > 0.1$. This is larger than the cleanup threshold at u, which is below 0.01. Hence, cleanup cannot terminate before pushing u.

Pushing u costs $\Theta(d(u)) = \Theta(m) = \Theta(1/\epsilon)$ work. Thus every insertion of (u, v) forces $\Omega(1/\epsilon)$ work.

Finally, take the update sequence that repeatedly toggles the edge (u, v). After every deletion and cleanup, the graph is again the initial graph, so the same argument applies to the next insertion. Therefore, the total work is $\Omega(k/\epsilon)$, and the amortized update time is $\Omega(1/\epsilon)$. □

B Exact Dynamic Algorithm For the Linear Case

In this section, we focus on the linear case, which is closely related to the PageRank problem. We seek an algorithm that computes the exact solution instead of an approximation. As before, we assume that the graph contains permanent self-loops. However, we do not require the graph to be unweighted, and we define the degree of a node as the sum of the weights of its incident edges. Suppose that each f_i is the identity function. Then the unique fixed point of the system is

$$z^* = [I - (1-\alpha)W]^{-1} \alpha s. \tag{5}$$

For a static graph, one can compute z^* exactly by computing the matrix inverse $[I - (1-\alpha)W]^{-1}$ in $O(n^\omega)$ time, where ω is the matrix multiplication exponent [BH74]. For a dynamic graph, there is an extensive line of work on dynamic matrix inverse and dynamic systems of linear equations. We adapt these results to the PageRank model, and in particular to the normalized weight matrix. Our algorithms are mainly inspired by the dynamic algorithm for systems of linear equations of [San04]. However, their algorithm does not directly give a dynamic algorithm for computing z^* in our setting, because an edge insertion or deletion may change up to two rows and two columns of W due to the normalized weights. Their algorithm does not directly support such updates. One could try to decompose the changes caused by an insertion or deletion into multiple operations, but this requires care, since their algorithm assumes that the matrix remains non-singular after each intermediate update.

In the following lemma, we show that an edge update induces a low-rank update to W. This directly follows from the fact that only two rows and two columns can change. However, the specific structure of the update is what allows us to obtain an efficient algorithm.

Lemma B.1. *Suppose an edge between u, v with weight δ is inserted or deleted from the graph, and let W_+ be the normalized weight matrix of the new graph. Then*

$$W_+ = W + UV^T$$

for some $U, V \in \mathbb{R}^{n \times 4}$. In other words, the edge update induces an update of rank at most 4 to W.

Proof. We use the subscript $+$ to denote quantities after the update. Let the signed change be $\eta = \delta$ for an insertion and $\eta = -\delta$ for a deletion. Let $E = [e_u \quad e_v]$. Then $A_+ = A + \eta(e_u e_v^T + e_v e_u^T)$ and $D_+ = D + \eta(e_u e_u^T + e_v e_v^T)$. Let $W = D^{-\beta} A D^{\beta-1}$ and $W_+ = D_+^{-\beta} A_+ D_+^{\beta-1}$.

Since only the degrees of u and v change, there are diagonal matrices $\Lambda, \Gamma \in \mathbb{R}^{2\times 2}$ such that $D_+^{-\beta} D^{\beta} = I + E\Lambda E^T$ and $D^{1-\beta} D_+^{\beta-1} = I + E\Gamma E^T$. For $i \in \{u, v\}$, these matrices have entries $\Lambda_{ii} = (d_i/d_{i,+})^{\beta} - 1$ and $\Gamma_{ii} = (d_{i,+}/d_i)^{\beta-1} - 1$.

Define $H \in \mathbb{R}^{2\times 2}$ by $EHE^T = \eta D_+^{-\beta}(e_u e_v^T + e_v e_u^T) D_+^{\beta-1}$. Equivalently, $H_{uv} = \eta d_{u,+}^{-\beta} d_{v,+}^{\beta-1}$, $H_{vu} = \eta d_{v,+}^{-\beta} d_{u,+}^{\beta-1}$, and the diagonal entries of H are zero.

Using these definitions,

$$W_+ = D_+^{-\beta} A_+ D_+^{\beta-1} = (I + E\Lambda E^T) W (I + E\Gamma E^T) + EHE^T.$$

Thus

$$W_+ - W = E\Lambda E^T W (I + E\Gamma E^T) + WE\Gamma E^T + EHE^T.$$

Now set

$$U = [E \quad WE\Gamma] \in \mathbb{R}^{n\times 4}$$

and

$$V^T = \begin{bmatrix} \Lambda E^T W (I + E\Gamma E^T) + HE^T \\ E^T \end{bmatrix} \in \mathbb{R}^{4\times n}.$$

Then $W_+ = W + UV^T$, so the update has rank at most 4. □

Next, we describe at a high level how z^* can be maintained under updates. To do this, we use the Woodbury matrix identity [Woo50], which gives a formula for the inverse of a matrix under a low-rank update.

Let $B := I - (1-\alpha)W$. Following Sankowski [San04], we maintain B^{-1} in an implicit form. Specifically, after k updates, we write $B = B_0 + \mathcal{U}\mathcal{V}^T$, where $\mathcal{U}, \mathcal{V} \in \mathbb{R}^{n\times 4k}$ and B_0 is the last rebuilt matrix. We maintain $M := B_0^{-1}$, $Y := M\mathcal{U}$, and the small inverse $G := (I_{4k} + \mathcal{V}^T Y)^{-1}$. We also maintain the current solution $z^* = B^{-1}\alpha s$. By the Woodbury identity,

$$B^{-1} = M - YG\mathcal{V}^T M. \tag{6}$$

We use the subscript $+$ to denote the state after a single edge update. Consider an update from W to W_+. By Theorem B.1, $W_+ = W + U_W V_W^T$ for some $U_W, V_W \in \mathbb{R}^{n\times 4}$. Then $B_+ = B + UV^T$, where $U = U_W$ and $V = -(1-\alpha)V_W$. Define $Z := B^{-1}U$ and $S := I_4 + V^T Z$. Applying the Woodbury identity again gives $B_+^{-1} = B^{-1} - ZS^{-1}V^T B^{-1}$. Since S has size 4×4, its inverse can be computed in $O(1)$ time. The new solution is therefore $z_+^* = z^* - ZS^{-1}V^T z^*$. Once Z is known, this update takes $O(n)$ time. We give the details for computing Z in the lemma below.

Lemma B.2. *Let an edge update produce $B_+ = B + UV^T$ with $U, V \in \mathbb{R}^{n\times 4}$. Then the maintained state and the solution $z_+^* = B_+^{-1}\alpha s$ can be updated in time $O(nk + k^2)$.*

Proof. Before the update, we maintain $B = B_0 + \mathcal{U}\mathcal{V}^T$, where $\mathcal{U}, \mathcal{V} \in \mathbb{R}^{n\times 4k}$, $M = B_0^{-1}$, $Y = M\mathcal{U}$, $G = (I + \mathcal{V}^T Y)^{-1}$, and $z^* = B^{-1}\alpha s$. Thus

$$B^{-1} = M - YG\mathcal{V}^T M.$$

By Theorem B.1, the edge update has $U = [E \quad WE\Gamma]$, where $E = [e_u \quad e_v]$ and $\Gamma \in \mathbb{R}^{2\times 2}$ is diagonal. We first compute $Z = B^{-1}U$. Since $B = I - (1-\alpha)W$, $B^{-1}W = \frac{1}{1-\alpha}(B^{-1} - I)$. Therefore,

$$Z = \left[B^{-1}E \quad \tfrac{1}{1-\alpha}(B^{-1}E - E)\Gamma\right].$$

It remains to compute $B^{-1}E$. Using the maintained inverse representation,

$$B^{-1}E = ME - YG\mathcal{V}^T ME.$$

The matrix ME consists of two columns of M, so it can be obtained in $O(n)$ time. Computing $\mathcal{V}^T ME$ costs $O(nk)$, multiplying by G costs $O(k^2)$, and multiplying by Y costs $O(nk)$. Hence Z can be computed in $O(nk + k^2)$ time.

Now define $S = I_4 + V^T Z$. Since S is 4×4, computing S^{-1} takes $O(1)$ time. By the Woodbury identity,

$$B_+^{-1} = B^{-1} - ZS^{-1}V^T B^{-1}.$$

Therefore, $z_+^* = z^* - ZS^{-1}V^T z^*$. The product $V^T z^*$ costs $O(n)$ time, and the remaining operations also cost $O(n)$ time. This is within the claimed bound.

It remains to update the maintained state. We append the new factors: $\mathcal{U}_+ = [\mathcal{U} \quad U]$ and $\mathcal{V}_+ = [\mathcal{V} \quad V]$. We also append MU to Y. The first two columns, ME, are already available. For the remaining two columns, $MWE\Gamma$, use $B_0 = I - (1-\alpha)W_0$ and $B = B_0 + \mathcal{U}\mathcal{V}^T = I - (1-\alpha)W$. These identities imply $W = W_0 - \frac{1}{1-\alpha}\mathcal{U}\mathcal{V}^T$. Thus

$$MWE = \frac{1}{1-\alpha}(ME - E) - \frac{1}{1-\alpha}Y\mathcal{V}^T E.$$

Therefore,

$$MU = \left[ME \quad \left(\tfrac{1}{1-\alpha}(ME - E) - \tfrac{1}{1-\alpha}Y\mathcal{V}^T E\right)\Gamma\right].$$

This costs $O(nk)$ time to compute, so $Y_+ = [Y \quad MU]$ can be updated in $O(nk)$ time.

Finally, we update $G_+ = (I_{4(k+1)} + \mathcal{V}_+^T Y_+)^{-1}$. After appending the new factors, we have $\mathcal{U}_+ = [\mathcal{U} \quad U], \mathcal{V}_+ = [\mathcal{V} \quad V], Y_+ = M\mathcal{U}_+ = [Y \quad MU]$. Therefore,

$$I_{4(k+1)} + \mathcal{V}_+^T Y_+ = \begin{bmatrix} I_{4k} + \mathcal{V}^T Y & \mathcal{V}^T MU \\ V^T Y & I_4 + V^T MU \end{bmatrix}.$$

Since $G = (I_{4k} + \mathcal{V}^T Y)^{-1}$ is already stored, we only need to invert this block matrix. Define

$$C := \mathcal{V}^T MU \in \mathbb{R}^{4k\times 4}, \qquad R := V^T Y \in \mathbb{R}^{4\times 4k}, \qquad Q := I_4 + V^T MU \in \mathbb{R}^{4\times 4}.$$

Then

$$I_{4(k+1)} + \mathcal{V}_+^T Y_+ = \begin{bmatrix} G^{-1} & C \\ R & Q \end{bmatrix}.$$

Let $\Sigma := Q - RGC \in \mathbb{R}^{4\times 4}$ be the Schur complement. By the block inverse formula,

$$G_+ = \begin{bmatrix} G + GC\Sigma^{-1}RG & -GC\Sigma^{-1} \\ -\Sigma^{-1}RG & \Sigma^{-1} \end{bmatrix}.$$

It remains to bound the cost of forming this expression. The matrix MU has already been computed. Forming $C = \mathcal{V}^T MU$ costs $O(nk)$ time, forming $R = V^T Y$ costs $O(nk)$ time, and forming $Q = I_4 + V^T MU$ costs $O(n)$ time. Computing GC and RG costs $O(k^2)$ time, since G has size $4k \times 4k$. The matrix Σ has size 4×4, so computing Σ^{-1} costs $O(1)$ time. Hence updating G_+ costs $O(nk + k^2)$ time.

Thus both the maintained state and the solution $z_+^* = B_+^{-1}\alpha s$ can be updated in $O(nk+k^2)$ time. $\square$

The above lemma gives an efficient algorithm as long as k is small. Thus, as k grows, we need a way to propagate all changes into B and make $\mathcal{U}, \mathcal{V}$ small again. The following lemma gives this rebuilding step. We use the rebuilding algorithm of Sankowski [San04, Theorem 2], but state it here explicitly to avoid ambiguity. The theorem and proof of Sankowski [San04] are stated for the case where $O(n^\gamma)$ columns of the original matrix have changed. We note that the proof extends directly to the case where $O(n^\gamma)$ columns and $O(n^\gamma)$ rows have changed. We state this form in the following lemma.

Lemma B.3. *Let A be nonsingular, and suppose $A' = A + \Delta$ is also nonsingular. Assume that Δ is supported only on rows and columns indexed by a set S, where $|S| = k = O(n^\gamma)$. Then A^{-1} can be updated to $(A')^{-1}$ using $O\left(n^{\omega(1,\gamma,1)}\right)$ arithmetic operations.*

Proof. Let $E \in \mathbb{R}^{n\times k}$ be the matrix whose columns are the standard basis vectors indexed by S. Since Δ is supported on $S \times [n] \cup [n] \times S$, we can write $\Delta = ER + CE^T$, where $R = E^T\Delta$ contains the changed rows, and $C = (I - EE^T)\Delta E$ contains the changed columns outside those rows.

Thus $A' = A + ER + CE^T = A + UV^T$, where

$$U = [E \quad C], \qquad V^T = \begin{bmatrix} R \\ E^T \end{bmatrix}.$$

Hence $U, V \in \mathbb{R}^{n \times 2k}$, so the update has rank at most $2k$. By the Woodbury identity,

$$(A')^{-1} = A^{-1} - A^{-1}U\left(I_{2k} + V^T A^{-1} U\right)^{-1} V^T A^{-1}.$$

The products $A^{-1}U$ and $V^T A^{-1}$ are rectangular products with one dimension $O(k) = O(n^\gamma)$, so they cost $O\left(n^{\omega(1,\gamma,1)}\right)$ arithmetic operations. The middle matrix has size $O(k) \times O(k)$, and the final product has dimensions $n \times O(k)$ by $O(k) \times n$. These steps are within the same bound. Therefore the full update costs $O\left(n^{\omega(1,\gamma,1)}\right)$ arithmetic operations. □

Using the above lemma, we rebuild after n^γ updates.

Corollary B.4. *After n^γ edge updates, the inverse B^{-1} can be rebuilt in $O\left(n^{\omega(1,\gamma,1)}\right)$ arithmetic operations.*

Proof. Each edge update changes at most two rows and two columns. Therefore, after n^γ edge updates, the total change is supported on $O(n^\gamma)$ rows and $O(n^\gamma)$ columns. By Theorem B.3, we can compute the updated inverse B^{-1} in $O\left(n^{\omega(1,\gamma,1)}\right)$ arithmetic operations. □

Combining the above results gives the following theorem.

Theorem B.5. *The fixed point z^* can be maintained by an algorithm with $O(n^\omega)$ initialization cost and $O\left(n^{\omega(1,\gamma,1)-\gamma} + n^{1+\gamma}\right)$ worst-case update time. The algorithm maintains z^* explicitly after each update. In particular, setting $\gamma \approx 0.5275$ gives update time $O\left(n^{1.5275}\right)$ under the current known bounds.*

Proof. We begin by computing $M = B^{-1}$ in $O(n^\omega)$ time. We then process updates using the lazy representation and rebuild after $O(n^\gamma)$ updates. By Theorem B.2, each lazy update costs $O\left(n^{1+\gamma}\right)$ time. By Theorem B.4, each rebuild costs $O(n^{\omega(1,\gamma,1)})$ time. This gives the stated bound in amortized time.

To obtain the same bound in worst-case time, we use the standard deamortization technique noted by Sankowski [San04]. We maintain two copies of the data structure. The active copy answers queries and processes updates using the lazy Woodbury representation. In parallel, the second copy performs the rebuild incrementally over the next n^γ updates. By spreading the $O\left(n^{\omega(1,\gamma,1)}\right)$ rebuild work evenly over these updates, the rebuild contributes $O\left(n^{\omega(1,\gamma,1)-\gamma}\right)$ worst-case time per update. At the end of the phase, the rebuilt copy becomes active. Using the matrix multiplication bounds of Alman et al. [Alm+25] and the interpolation method for $\omega(1, \gamma, 1)$ from Van Den Brand, Nanongkai, and Saranurak [VNS19], we optimize over γ and obtain $\gamma \approx 0.5275$, which gives the claimed bound. □

C Experiment Details of Section 4.1

This section describes the experimental setup for Table 1. We provide the general setup, the hyperparameter search and the configurations selected for each cell of Table 1 in Appendix C.1, and a discussion of the CHAMELEON and SQUIRREL datasets in Appendix C.2.

Datasets. We consider several common datasets. All datasets contain feature vectors for all nodes, and labels for a subset of nodes are provided in the training set. CORA, CITESEER [Sen+08], and PUBMED [Nam+12] are citation graphs, where nodes are papers and edges are citations. ACTOR [Tan+09; Pei+20] is a graph of actors, where edges denote co-occurrence on the same Wikipedia page. WebKB [Cra+98] is a dataset collected from university websites where nodes represent webpages, and edges represent links between them. We use the CORNELL, WISCONSIN, and TEXAS graphs from this dataset. CHAMELEON and SQUIRREL are graphs of Wikipedia pages on specific topics [RAS21; Pei+20], where nodes are pages and edges are mutual links between them. For a discussion of CHAMELEON and SQUIRREL, see Appendix C.2.

Splits. All datasets are loaded through PyTorch Geometric [FL19]. CORA, CITESEER, and PUBMED use the standard Planetoid train/validation/test split [YCS16]. CORNELL, TEXAS, WISCONSIN (WebKB), ACTOR, CHAMELEON, and SQUIRREL use the ten random $48\%/32\%/20\%$ splits released by Pei et al. [Pei+20]. We make every graph undirected and standardize each feature column to have zero mean and unit variance. Table 3 shows the scales of the datasets used in these experiments.

The PyTorch Geometric Planetoid distributions of CORA, CITESEER, and PUBMED are MIT licensed, with the underlying PUBMED records also subject to the NLM Data License. The WebKB graphs and ACTOR are loaded through PyTorch Geometric, but their standalone dataset licenses appear unspecified. CHAMELEON and SQUIRREL, including the filtered versions released by Platonov et al. [Pla+23], are distributed under the MIT license.

Method. Following Zheng et al. [Zhe+22], we first propagate the features using the PageRank model and then train an MLP on the propagated features using the training labels. For the non-linear model, we use the same activation function for all node functions f_i. In addition to the propagated features z, we pass the pre-activation vectors $y = \alpha s + (1-\alpha)Wz$ to the MLP. When the activation function is the identity, $y \approx z$.

Baselines. We ran every baseline in Table 1 ourselves, on exactly the same splits as our model: GCN [KW17], GAT [Vel+17], Geom-GCN [Pei+20], APPNP [GBG18], ACM-GCN [Lua+22], and GREAD [Cho+23]. When the authors of a method published a configuration for a dataset and split, we used it. Otherwise, we tuned the method over at most 288 configurations, the same size our model uses for each activation function. The tuning has two stages. We first rank the configurations by validation accuracy on one seed. We then rerun the best configurations on five new seeds, select the one with the highest validation accuracy, and report its test accuracy from these runs. We report standard errors in the same way as for our model.

Table 3: Statistics for datasets used in Section 4.1.

Dataset	Nodes (n)	Edges	Feature dimension	Classes
CORNELL	183	164	1,703	5
TEXAS	183	167	1,703	5
WISCONSIN	251	231	1,703	5
SQUIRREL	5,201	159,248	2,089	5
ACTOR	7,600	14,992	932	5
CHAMELEON	2,277	23,370	2,325	5
CORA	2,708	5,278	1,433	7
CITESEER	3,327	4,552	3,703	6
PUBMED	19,717	44,324	500	3
CHAMELEON (FILTERED)	890	8,854	2,325	5
SQUIRREL (FILTERED)	2,223	46,998	2,089	5

Propagation. After normalization, we multiply the features by a feature scale c_{feat} before propagation. The classifier receives the concatenation $[y, z]$, where $y = \alpha s + (1-\alpha)Wz$ is the pre-activation vector. We include c_{feat} in the search grid because it interacts with the saturation thresholds of activations such as tanh.

Activations. We evaluate the linear baseline (denoted "Linear"), $\tanh$, $\mathrm{sigmoid}$, scaled tanh with $c \in \{0.5, 1, 1.5, 2, 3, 4, 5, 6, 8\}$, hard tanh with $c \in \{0.3, 0.5, 0.7, 1, 1.5, 2, 2.5\}$, and shifted tanh with $c \in \{0.5, -0.5, -0.9, -1.2\}$. This broad set lets us test activations with different shapes and saturation behavior. Table 1 reports only the activations that achieve the best test accuracy in at least one dataset column; the remaining activations were also evaluated.

Classifier. The classifier is either a linear layer or a one-hidden-layer MLP with hidden width 512, ReLU, and dropout 0.5. The classifier type is selected as part of the search grid. Both classifiers are trained with Adam on the cross-entropy loss, with weight decay 5×10^{-4}, for at most 400 epochs. We use early stopping after 100 epochs without improvement in validation accuracy. We report the test accuracy at the epoch with the highest validation accuracy.

Compute. Each run for an (activation, dataset, seed, split, hyperparameter) configuration uses one NVIDIA RTX A4000 GPU, 4GB RAM, and 2 CPU cores and takes less than 5 seconds to complete.

C.1 Hyperparameter Sweep and Selected Configurations

Table 4: Hyperparameter search grid used for each activation–dataset pair.

Hyperparameter	Values
α	$\{0.01, 0.1, 0.2, 0.3, 0.5, 0.7\}$
β	$\{0.5, 1.0\}$
c_{feat}	$\{1, 3, 5\}$
Classifier	{linear, MLP}
Learning rate	$\{10^{-3}, 10^{-2}, 10^{-1}, 0.5\}$
Weight decay	5×10^{-4}
MLP hidden dimension	512
MLP dropout	0.5
Training epochs	400
Early stopping patience	100
Optimizer	Adam
Loss	Cross-entropy

We use the same search grid in Table 4 for every (activation, dataset) pair. For each (activation, dataset) cell, we evaluate every grid point on 5 random seeds for each of the 10 splits described above, for a total of 50 runs per configuration. We then select the grid point with the highest mean validation accuracy across these runs. In Table 1, we report the mean and standard error of the test accuracy for the selected grid point. The selected configurations are listed in Table 5.

Table 5: Configuration of $(\alpha, \beta, c_{\text{feat}}, \text{classifier}, \text{lr})$ selected by validation accuracy for each (activation, dataset) cell of Table 1. "L" denotes the linear classifier and "M" denotes the MLP.

Activation	CORN.	TEXAS	WISC.	SQUI.	ACTOR	CHAM.	CORA	CITE.	PUBM.
Linear	0.7/1/1/L/0.5	0.7/1/1/L/0.5	0.7/0.5/1/M/0.1	0.01/0.5/3/M/0.001	0.7/0.5/1/M/0.5	0.1/1/5/M/0.01	0.01/1/3/M/0.5	0.1/1/1/L/0.1	0.01/0.5/3/M/0.1
Sc Tanh 0.5	0.7/1/5/M/0.01	0.7/0.5/5/M/0.1	0.7/0.5/3/M/0.01	0.01/0.5/3/M/0.001	0.7/1/3/M/0.001	0.1/1/3/M/0.01	0.01/0.5/5/L/0.01	0.1/0.5/1/L/0.1	0.1/1/1/L/0.5
Sc Tanh 8	0.7/1/1/M/0.1	0.5/1/1/M/0.1	0.2/0.5/5/M/0.01	0.01/0.5/3/M/0.01	0.5/1/5/M/0.001	0.01/1/1/M/0.01	0.01/0.5/1/M/0.01	0.01/0.5/3/M/0.01	0.01/0.5/1/M/0.01
Sh Tanh -1.2	0.5/1/3/M/0.01	0.3/0.5/3/M/0.01	0.3/0.5/5/M/0.01	0.5/1/1/M/0.001	0.3/1/1/M/0.001	0.7/1/1/L/0.01	0.5/1/5/M/0.01	0.7/0.5/5/M/0.01	0.1/0.5/1/L/0.1
H Tanh 2.5	0.7/1/5/M/0.01	0.7/1/5/M/0.01	0.7/0.5/3/M/0.01	0.01/0.5/3/M/0.001	0.7/1/3/M/0.001	0.1/1/3/M/0.01	0.1/1/5/M/0.5	0.1/0.5/1/L/0.1	0.1/1/3/L/0.5

C.2 CHAMELEON and SQUIRREL

CHAMELEON and SQUIRREL [RAS21] are Wikipedia link networks that are widely used as heterophilous benchmarks. Platonov et al. [Pla+23] show that the publicly distributed versions of these datasets contain duplicated nodes, which can leak across the train, validation, and test splits. This leakage can increase the accuracy of methods that effectively memorize duplicated nodes. They release filtered versions with the duplicated nodes removed. The results in Table 1 use the *original* Geom-GCN preprocessed splits, which are standard for these datasets. Table 6 reports our model and the baselines on the filtered versions, using the ten splits of Platonov et al. [Pla+23]. For our model, we use the same hyperparameter search grid as in Appendix C.1, and Table 7 shows the selected hyperparameters. The baselines are tuned as described in Appendix C. We do not include Geom-GCN, because its authors released the data it needs only for the original graphs.

Table 6: Test accuracy on the filtered CHAMELEON and SQUIRREL datasets of Platonov et al. [Pla+23]. The best method is underlined.

Method	CHAM. (FILT.)	SQUI. (FILT.)
GCN	35.6 ± 0.8	34.8 ± 0.4
GAT	35.3 ± 0.8	34.9 ± 0.4
APPNP	36.1 ± 0.9	34.9 ± 0.4
ACM-GCN	38.9 ± 0.7	35.9 ± 0.5
GREAD	44.0 ± 1.0	41.0 ± 0.5
Linear	41.1 ± 1.1	40.4 ± 0.6
Sh Tanh -0.5	$\underline{45.4 \pm 1.2}$	44.0 ± 0.5
Sh Tanh -1.2	43.3 ± 0.9	$\underline{44.6 \pm 0.6}$

Table 7: Configuration of $(\alpha, \beta, c_{\text{feat}}, \text{classifier}, \text{lr})$ selected by validation accuracy for each (activation, dataset) cell of Table 6. "L" denotes the linear classifier and "M" denotes the MLP.

Activation	CHAM. (FILT.)	SQUI. (FILT.)
Linear	0.01/0.5/1/M/0.1	0.01/0.5/1/M/0.1
Sh Tanh -0.5	0.7/0.5/3/M/0.001	0.5/0.5/1/M/0.01
Sh Tanh -1.2	0.5/0.5/5/M/0.01	0.7/0.5/1/M/0.001

D Experiment Details of Section 4.2

Datasets and snapshots. We use two graphs from the Open Graph Benchmark [Hu+20]: `arxiv`, a citation network, and `products`, the Amazon products co-purchasing network. We also use the synthetic `SBM-500K` dataset generated by Zheng et al. [Zhe+22]. The edge updates for `arxiv` and `products` are incremental edge additions, while `SBM-500K` includes both edge additions and edge deletions. The `arxiv` and `products` benchmarks are used with their official train/validation/test splits. We follow the snapshot split protocol of Zheng et al. [Zhe+22]. First, we make the graph undirected and standardize every feature column to have zero mean and unit variance. We then remove every non-self edge incident to a training node. The remaining edges form the initial graph, and the removed edges form the insertion stream. The insertion stream is shuffled with constant random seed and split into contiguous snapshots. We use 16 snapshots for `arxiv`, 15 snapshots for `products`, and `SBM-500K` contains 10 snapshots. Self-loops are present throughout the experiment and are not part of the update stream.

For `SBM-500K`, the dataset is generated with the InstantGNN [Zhe+22] SBM generator[4] using $500{,}000$ nodes, 50 communities, average in-community degree 20, average out-community degree 1, 10 snapshots, and $2{,}500$ changed nodes per snapshot. The dataset provides initial labels and per-snapshot labels. Since features and splits are not shipped with the text edge stream, we generate fixed-seed sparse random features with 256 dimensions and 16 nonzero entries per node, standardize every feature column, and use random train/validation/test splits of $70\%/20\%/10\%$.

Under this construction, `arxiv` has $169{,}343$ nodes, 128 features, $213{,}848$ initial undirected non-self edges, and $1{,}157{,}799$ total undirected non-self edges after all insertions. The `products` graph has $2{,}449{,}029$ nodes, 100 features, $33{,}592{,}708$ initial undirected non-self edges, and $61{,}859{,}012$ total undirected non-self edges after all insertions. The `SBM-500K` graph has $500{,}000$ nodes, 256 generated features, 50 classes, $6{,}820{,}831$ initial undirected non-self edges, and $6{,}823{,}982$ final undirected non-self edges after all updates. Across its 10 snapshots, the stream contains $1{,}263{,}739$ unique undirected edge-update events, consisting of $633{,}445$ insertions and $630{,}294$ deletions.

Dataset licenses. The Open Graph Benchmark lists `arxiv` under ODC-BY and `products` under the Amazon License. The `SBM` datasets are synthetic datasets generated using the code [4] from InstantGNN [Zhe+22]; the repository does not specify a license for these datasets.

[4] `https://github.com/zheng-yp/InstantGNN`

Propagation methods. For both the linear and non-linear models, we use $\beta = 0.5,\ \epsilon = 10^{-7}$. We use $\alpha = 0.1$ for `arxiv` and `products`, and $\alpha = 0.001$ for `SBM-500K`.

Before propagation, we multiply the standardized input features by 3. The linear model uses the identity activation. The non-linear model uses hard tanh, $f(x) = \min(2.5, \max(-2.5, x))$. We use the same activation for every node and every feature dimension.

We compare three propagation strategies. The "scratch" method applies all edge updates in the current snapshot, resets the propagation state to the canonical initialization, and runs cleanup from scratch. The "batched" method applies all insertions in the current snapshot as one batch and then runs cleanup once. The "single" method applies insertions one edge at a time, runs the dynamic update after each edge, and returns the maintained representation after every edge insertion; for reporting, we evaluate the representation after the last edge of each snapshot. Thus, "single" solves the strictly stronger online problem, while "batched" matches the snapshot update used in Zheng et al. [Zhe+22].

Propagation is implemented in C++. The implementation stores the feature, propagated, pre-activation, and residual arrays as dense double-precision arrays of shape $F \times n$. Cleanup is run independently for each feature dimension, and dimensions are parallelized with OpenMP. We use the same approximation guarantees as Zheng et al. [Zhe+22].

Classifier and parameters. After each initial propagation or snapshot update, we train a classifier from scratch on the current propagated features. The classifier input is the concatenation $[z, y]$, where $y = \alpha s + (1-\alpha)Wz$ is the pre-activation vector. For `arxiv` and `products`, the classifier is a 4-layer MLP with hidden width 1024, batch normalization after hidden linear layers, ReLU activations, and Adam optimization with learning rate 10^{-4} and no weight decay. For `SBM-500K`, we use a 2-layer MLP with hidden width 1024, dropout 0.1, Adam learning rate 0.01, and no weight decay. We train `arxiv` and `products` for at most 1000 epochs and `SBM-500K` for at most 200 epochs, with early stopping after 50 epochs without validation improvement. We use dropout 0.3 on `arxiv`, dropout 0.5 on `products`, and dropout 0.1 on `SBM-500K`. The `arxiv` classifier is trained full-batch, while `products` uses mini-batches of size 512 and `SBM-500K` uses mini-batches of size 1024.

These parameters were all inherited from experiments of Zheng et al. [Zhe+22].

Compute. Each dynamic experiment on `arxiv`, `products`, and `SBM-500K` is run with one NVIDIA RTX A4000 GPU, 16 CPU cores, and 32GB RAM. The runs for Table 8 use 16 CPU cores and no GPU. They need about 9GB of RAM on `SBM-500K`, 32GB on `SBM-2M`, 77GB on `SBM-5M`, and 700GB on `papers100M`.

Reported quantities. The accuracy plots report the test accuracy of the classifier trained on the propagated features available at each snapshot. The cumulative-time plots report the cumulative propagation time, including the initial propagation and all processed snapshots. The per-snapshot time plots report only propagation time for the corresponding initial graph or snapshot update; classifier training time is excluded from these timing plots.

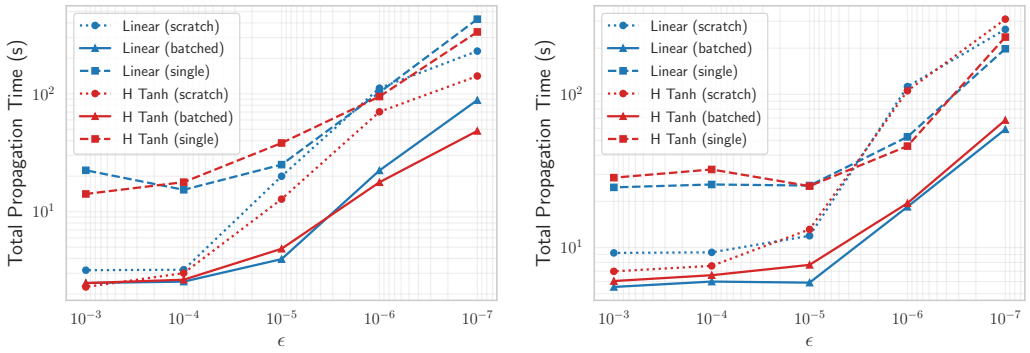
Larger graphs. We also run two larger versions of the SBM benchmark, `SBM-2M` and `SBM-5M`, and the `papers100M` graph from Open Graph Benchmark [Hu+20], which has 111M nodes and 1.6B edges. Table 8 reports the propagation time per snapshot. The cost of recomputation grows with the size of the graph, while the cost of dynamic updates does not. We generate `SBM-2M` and `SBM-5M` with the same InstantGNN generator as `SBM-500K`, changing only the number of nodes. We scale the number of communities so that each community has $10{,}000$ nodes, and keep the degrees, the 10 snapshots, and the $2{,}500$ changed nodes per snapshot. So each snapshot has about $126{,}000$ edge updates at every size. We use the same α, β, and ϵ as for `SBM-500K`. For `papers100M`, we follow the same snapshot split protocol as for `arxiv` and `products`, with 20 snapshots of about 2M edge insertions each. We use $\alpha = 0.1$, $\beta = 0.5$, and $\epsilon = 2.2 \times 10^{-9}$. We run each SBM setting with 5 random seeds and `papers100M` once.

Varying ϵ and α. To study how the parameters affect running time, we repeat the same experiment while varying ϵ and α on the `arxiv` and `SBM-500K` datasets. Figures 4 and 5 show the results. We find that the observed running times are consistent with the theoretical dependence on these parameters. For Figures 4 and 5, we vary $\epsilon \in \{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and

Table 8: Propagation time per snapshot, in seconds, on larger graphs.

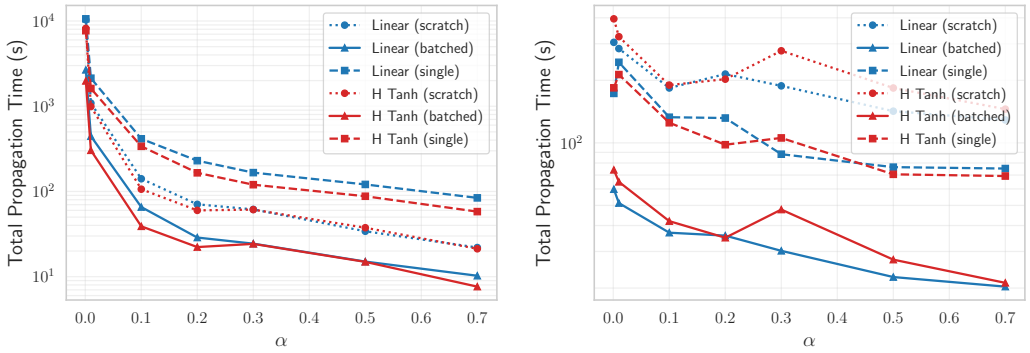
		Hard tanh			Linear		
Dataset	Nodes	scratch	single	batched	scratch	single	batched
SBM-500K	0.5M	34.9	19.0	4.3	27.2	20.0	4.3
SBM-2M	2M	68.3	7.3	1.3	55.2	4.6	1.3
SBM-5M	5M	119.1	3.8	1.3	89.2	3.8	1.0
papers100M	111M	20,248	579.7	116.6	31,695	777.9	151.3

$\alpha \in \{0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 0.7\}$, respectively. We run each setting with two random seeds and report the mean over the two runs.



(a) arxiv (b) SBM-500K

Figure 4: Total propagation time of varying ϵ values. The propagation time scales inversely with ϵ.



(a) arxiv (b) SBM-500K

Figure 5: Total propagation time of varying α values. Our theoretical bounds show that the propagation time scales with $1/\alpha^2$.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the intended contribution. The claims are tied to the assumptions and guarantees developed in the theory sections and to the empirical comparisons.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include limitations of our work and future directions in the conclusion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The theoretical assumptions are stated in the model and algorithm sections. Formal proofs are provided in the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The experiments section and the appendix describe the datasets, baselines, evaluation goals, the data splits, preprocessing, dynamic stream construction, hyperparameter grids, and selected configurations.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We publish our code in `https://github.com/mahdi-jfri/dynamic-nl-public-code`. All experiments can be reproduced by this code.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper and appendix specify the datasets, preprocessing, train/validation/test splits, dynamic update protocol, evaluated activations, hyperparameter grid, validation selection procedure, and selected configurations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The accuracy experiments report means and standard errors for selected configurations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We include compute details in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conform to all points in NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Our work is foundational research and there is no direct path to any negative applications.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk models and the experiments use established public graph benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We include license information for the datasets we use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not introduce or release a new asset.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The research does not involve crowdsourcing experiments or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The research does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development and experiments do not use LLMs as an important, original, or non-standard component.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.