GNNs on Evolving Graphs: A Benchmark of Incremental Updates and Meta-Learning Approaches

Sriram Srinivasan
Hamdan Alabsi
Rand Obeidat
Bowie State University
Bowie, MD, USA
ssrinivasan@bowiestate.edu
halabsi@bowiestate.edu
robeidat@bowiestate.edu

Sanjukta Bhowmick
University of North Texas
Denton, TX, USA
sanjukta.bhowmick@unt.edu

The Challenge of Dynamic Graphs

A traditional GNN operates by aggregating information from a node's neighbors to generate a new embedding. This process is typically applied to every node in the graph, making it a **full-graph inference** operation. When the graph changes, even slightly (e.g., a single node or edge is added), the embeddings for many connected nodes become stale. The naive solution is to re-run the full inference on the updated graph, but this does not scale for large, frequently changing graphs. The goal is to develop a method that provides accurate, up-to-date embeddings with minimal computational cost. In order to address this we propose two novel approaches standard incremental update, and Meta-learning approach.

The Standard Incremental Update Approach aka Zero Shot Learning

The most straightforward way to avoid full re-computation is to use an **Asynchronous incremental update**. Instead of re-running the entire model, we only update the parts of the graph that are directly affected by the change. This is based on the principle that a node's embedding is primarily influenced by its local neighborhood.

The k-Hop Subgraph Algorithm

This method isolates the affected area of the graph. The algorithm can be described as follows:

1. **Identify Changed Nodes:** Pinpoint the nodes that have been added, deleted, or whose features have been modified.

2. **Extract the Subgraph:** For each changed node, we identify its k-**hop neighborhood**—all nodes that are at most k edges away. We then combine these neighborhoods to form a single, small subgraph.

3. **Local Inference:** Run the pre-trained GNN model on this isolated subgraph. The model will compute new embeddings for all nodes within the subgraph.

4. **Update Embeddings:** The newly computed embeddings for the nodes in the subgraph are used to replace their old embeddings in the main graph's embedding matrix.

Algorithm 1: The k-Hop Subgraph Update Algorithm

Input: Pre-trained GNN model M, updated graph G', set of changed nodes C, hop count k
Output: Updated embedding matrix E'

for *each node* $v \in C$ **do**
  Extract k-hop neighborhood N_v from G';
end
Combine all neighborhoods to form subgraph S;
Compute new embeddings for nodes in S using M: $E_S = M(S, \text{features}_S)$;
Update embeddings in the full matrix: $E'[N_S] = E_S$;
return E';

This approach is much faster than full re-computation because it performs inference on a small subgraph, making the computational cost proportional to the subgraph's size, not the entire graph.

Meta-Learning Approach aka Few-Shot Learning

We introduce a meta-learning approach to train a GNN to be a "fast learner" for incremental tasks. Instead of training on a single, static graph, the model learns to specialize in performing small, localized updates, allowing it to act as an expert at efficient, incremental learning.

The Meta-Learning Training Algorithm

The training process for the meta-learned model is unique and focuses on training the GNN to produce embeddings for a local subgraph that closely match the embeddings it would produce if it had seen the entire, full graph. This is achieved by first computing a set of "ground truth" embeddings for the entire graph and then training the model to replicate these embeddings on a series of small, randomly sampled k-hop subgraphs. This process forces the model to learn the general principles of embedding propagation from limited local context.

The training process is as follows:

1. **Compute Ground Truth Embeddings:** First, we run a single, full-graph inference pass using a standard, untrained GNN. This gives us a set of **ground truth embeddings** for every node in the graph. These embeddings represent what an ideal GNN would produce on the complete graph, and they serve as the target for our meta-learner.

, . ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM

2. **Iterative Subgraph Training:** We then train our GNN model for a number of epochs on a series of small, randomly sampled k-hop subgraphs. In each training step, we perform the following:

- Sample a small subset of nodes to simulate a "change" to the graph.
- Extract the k-hop subgraph around these nodes.
- Pass this small subgraph through the model to get predicted embeddings.
- Compare the predicted embeddings to the pre-computed ground truth embeddings for the same nodes using a loss function (e.g., Mean Squared Error).
- Backpropagate the loss to update the model's weights.

This iterative training forces the model to learn a general "recipe" for embedding propagation from limited context. This makes the GNN robust, accurate, and suitable for efficient incremental updates on dynamic graphs.

Inference on Dynamic Graphs

Once the model has been trained using the meta-learning approach, it is ready to be used on a truly dynamic graph. During this phase, there is no further training.

When a graph evolves with new nodes and edges, the process is simple and efficient:

1. **Identify Changes:** The system detects which nodes have been added or which existing nodes have had their features or connections modified. These are the "changed nodes." 2. **Extract Subgraph:** The trained meta-learned GNN is used to extract the k-hop neighborhood of these changed nodes. This is the only part of the graph that the model needs to see. 3. **Update Embeddings:** The model performs a single, fast forward pass on this small subgraph to compute new embeddings. 4. **Integrate:** These new embeddings are used to update the corresponding nodes in the full graph's embedding matrix.

Because the model was trained to perform well on these specific types of localized tasks, it can produce highly accurate embeddings in milliseconds, using far less computational resources than a full re-computation. The model, now an expert at local inference, doesn't need to see the entire graph to understand how a change affects its local neighborhood.

Benchmarking and Comparison

To benchmark our approach, we put it to the test on a diverse range of real-world and synthetic networks, including RMAT, ER, and G models. We simulated a dynamic environment by randomly inserting 60,000 nodes and deleting 25,000, creating a rigorous test for our methods. All experiments were powered by an NVIDIA Tesla A100 GPU with 32GB of RAM, utilizing PyTorch GPU 2.5 and Python 3.12, ensuring consistent and high-performance execution.

The performance is measured using five key metrics:

- **Inference Time (Latency)**
- **Accuracy (MSE)**
- **Power Consumption**
- **Memory Footprint**
- **Throughput**

The results typically show that both incremental methods are significantly faster and efficient than full re-computation Figure 2. This demonstrates that by training a GNN to be a specialist at local inference, we can achieve the best of both worlds: the **speed of incremental updates** combined with the **accuracy of full re-computation**. Figure 1 presents the performance metric of both approaches when compared to the recomputation approach for PPI Network [4]. We observe that for all networks, both approaches are scalable Figure 2. Both approaches consume less power when compared to the recomputation approach. All methods have a similar memory footprint of about 5.6 GB. The Full Recomputation method has the highest throughput because it processes the entire graph. In contrast, the incremental methods have lower throughput but are more efficient, as they only update the necessary nodes. The meta-learned incremental approach is more accurate than the standard incremental approach because it adapts to shifts in data, making its accuracy closer to that of full retraining.

This research is inspired by foundational works on GNNs, such as the original GNN paper by Kipf and Welling [3], as well as more recent advancements in dynamic graph learning [2], and meta-learning for few-shot learning [1]. InkStream [5],is a specific, optimized method for efficiently performing an existing task: GNN inference on a dynamic graph. In this work model weights are static; the proposed approaches can adapt when model weights change.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1126–1135.

[2] H. Jin and G. Chen. 2020. A Joint-Learning-Based Dynamic Graph Learning Framework for Structured Prediction. *MDPI Sensors* 12, 11 (2020), 2357.

[3] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR '17)*.

[4] Damian Szklarczyk, Rebecca Kirsch, Mikaela Koutrouli, Katerina Nastou, Farrokh Mehryary, Radja Hachilif, Annika L. Gable, Tai Fang, Nadezhda T. Doncheva, Sampo Pyysalo, Peer Bork, Lars J. Jensen, and Christian von Mering. 2023. The STRING database in 2023: protein–protein association networks and functional enrichment analyses for any sequenced genome of interest. *Nucleic Acids Research* 51, D1 (2023), D638–D646. https://doi.org/10.1093/nar/gkac1091

[5] Dan Wu, Zhaoying Li, and Tulika Mitra. 2025. Inkstream: Instantaneous GNN Inference on Dynamic Graphs via Incremental Update. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1273–1285.

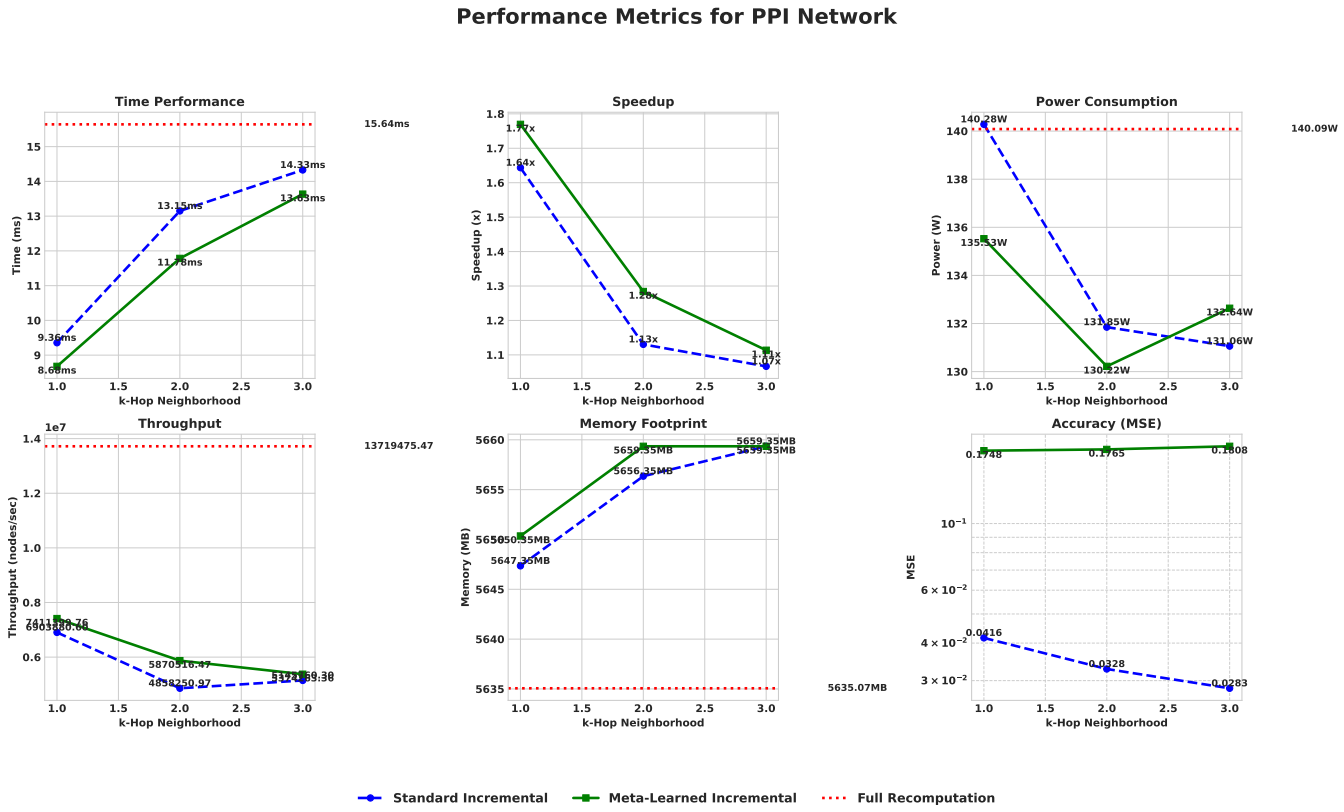

Figure 1: Performance Metrics for PPI Network

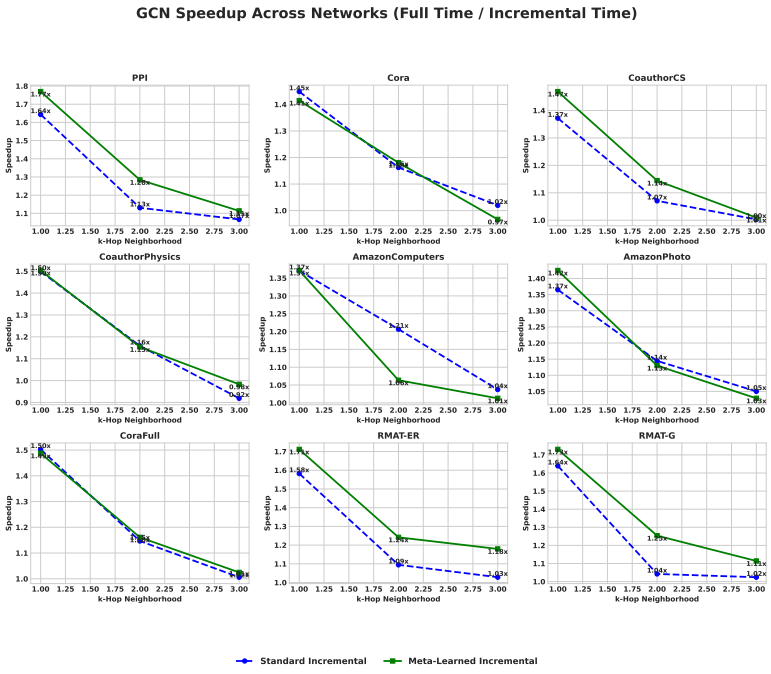

Figure 2: Speedup across Networks (Full Time / Incremental Time)