Design, Performance, and Hardware-Aware Evaluation of Graph Neural Networks for Hierarchical Influence Detection

Ashik Ali Shaik

December 14, 2025

Abstract

Influence detection in social networks is inherently relational: a user's importance can depend more on who they are connected to than on their standalone attributes. This report studies hierarchical influence detection, focusing on identifying "influencers of influencers"—nodes that may not have the largest direct following but exert strong indirect influence via connections to other influential nodes. We present a two-phase empirical framework: (i) a controlled synthetic benchmark designed to isolate multi-hop influence patterns that cannot be inferred reliably from node features alone, and (ii) an evaluation on real-world graphs (StackOverflow and WikiVote) to test whether the same advantages hold at realistic scale and noise. We compare multiple GNN architectures (GCN, GraphSAGE, GAT) against strong feature-only baselines (a neural MLP-style model and XGBoost) using accuracy, macro-F1, ROC/AUC, and confusion matrices, and we report learning dynamics via training curves. Finally, we benchmark CPU vs NVIDIA GPU performance for training and inference, including hop-sweep experiments that vary neighborhood depth. Across both phases, GNNs consistently capture hierarchical influence patterns that baselines miss, especially when the label signal is structural. GPU speedups grow with model depth and hop count, indicating that parallel sparse aggregation and memory bandwidth—rather than dense matrix throughput—dominate GNN acceleration. We complement quantitative results with an interactive graph visualizer to qualitatively validate influence paths and neighborhood structure.

Contents

1 Introduction

Social influence analysis underpins applications such as influencer marketing, recommendation, and information diffusion monitoring. A key difficulty is that influence can be *indirect.* For example, a user may have relatively few direct followers but can still be highly influential if their followers are themselves highly influential. This motivates the task studied here: detecting *influencers of influencers.*

Conventional feature-vector models (MLP, CNN on tabular inputs, gradient-boosted trees) operate on node attributes (e.g., follower count, post count) and cannot directly represent *who influences whom.* Graph Neural Networks (GNNs) address this by learning from both node features and graph connectivity through message passing. This report provides a comprehensive, end-to-end study—from dataset generation through model training, evaluation, qualitative visualization, and hardware-aware performance analysis.

1.1 Research questions

We structure the work around four questions:

RQ1: When the label depends on multi-hop structure, do GNNs outperform feature-only baselines?

RQ2: How do different GNN architectures compare (accuracy, F1, ROC/AUC, and error modes)?

RQ3: How does CPU vs GPU performance change with model depth (layers) and neighborhood depth (hops)?

RQ4: Which NVIDIA GPU architectural properties best explain the observed speedups for GNN workloads?

1.2 Contributions

- A synthetic hierarchical-influence benchmark that explicitly embeds "influencers of influencers" patterns while controlling feature confounders.
- A comparative evaluation of GNNs vs XGBoost and a feature-only neural baseline, using consistent metrics and visual diagnostics (confusion matrices, ROC curves, learning curves).
- A CPU/GPU benchmarking suite quantifying training and inference times, including hop-sweep experiments on real data.
- A graph visualizer and associated exporters that provide qualitative verification of structure and influence-path patterns.

2 Background: Graph Neural Networks from First Principles

2.1 Graphs, features, and supervised node classification

A graph is defined as $G = (V, E)$, where V is a set of nodes (users) and E is a set of edges (relations such as follows, replies, comments, votes). Each node $v \in V$ has a feature vector $\mathbf{x}_v \in \mathbb{R}^d$. In node classification, we aim to learn a function f such that $\hat{y}_v = f(G, \mathbf{X}, v)$ predicts a label y_v for each node.

Feature-only models approximate $\hat{y}_v \approx f(\mathbf{x}_v)$ and thus cannot condition on E. GNNs explicitly condition on neighborhoods defined by E.

2.2 Message passing: the core GNN mechanism

Most modern GNNs can be expressed via a *message passing neural network* (MPNN) update:

$$\mathbf{m}_v^{(k)} = \mathrm{AGG}\Big(\big\{\phi^{(k)}(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{e}_{uv}) : u \in \mathcal{N}(v)\big\}\Big), \tag{1}$$

$$\mathbf{h}_v^{(k)} = \psi^{(k)}\big(\mathbf{h}_v^{(k-1)}, \mathbf{m}_v^{(k)}\big), \tag{2}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $\mathcal{N}(v)$ is the neighbor set, $\mathbf{e}_{uv}$ are optional edge features, AGG is a permutation-invariant operator (sum/mean/max), and ϕ, ψ are learnable functions. Intuitively:

- **Local-to-global**: stacking K layers propagates information up to K hops.
- **Structure sensitivity**: the update depends on graph connectivity, enabling multi-hop influence patterns.
- **Sparse, irregular compute**: neighborhoods vary in size; operations are dominated by sparse aggregation.

2.3 Architectures used in this study

We focus on widely used GNN families:

- **GCN** [4]: normalized neighborhood averaging with linear transforms.
- **GraphSAGE** [3]: inductive aggregation (often mean) and the option to sample neighbors for scalability.
- **GAT** [6]: attention weights to reweight neighbors based on learned compatibility.

For baselines we use:

- **XGBoost** [1]: strong non-linear tabular baseline.
- **Feature-only neural baseline**: an MLP/CNN-style network operating on node features without access to edges.

3 Problem Definition: Hierarchical Influence Detection

We define **hierarchical influence detection** as a binary node classification task where the positive class includes:

1. **Influencers**: nodes that directly influence many others.
2. **Influencers of influencers**: nodes that may have few direct links, but connect to influencers, meaning their influence is amplified indirectly.

Crucially, influencers of influencers are characterized by *multi-hop* relational patterns rather than easily separable feature values. This creates a natural stress-test for GNNs versus feature-only methods.

4 Methodology

4.1 Overall experimental flow

The project follows a two-phase flow:

1. **Phase 1 (Synthetic)**: construct a controlled graph with explicit hierarchical influence patterns, train models, and analyze both accuracy and CPU/GPU runtime as model depth increases.

2. **Phase 2 (Real data)**: repeat comparisons on real graphs and run hop-sweep experiments to test how neighborhood depth affects both predictive performance and GPU speedups.

4.2 Phase 1: Synthetic dataset generation

The synthetic generator creates a graph with distinct structural roles while ensuring node features alone are insufficient. The key design is to embed patterns where a "meta-influencer" connects to several influencers, and those influencers connect to many regular users. In terms of influence diffusion, the meta-influencer has strong second-order reach even if its first-order degree is modest.

4.2.1 Node types and labels

We create three conceptual types:

- **Meta-influencer (influencer of influencers)**: positive label.
- **Influencer**: positive label.
- **Regular user**: negative label.

To avoid trivial separability, follower-like counts and activity-like features are designed to overlap across types (e.g., a meta-influencer may not have the largest raw follower count). Thus, the intended signal is the *graph structure*.

4.2.2 Synthetic generation code (representative snippet)

The full implementation is provided in the project scripts (Phase 1 generator + training). The central idea is illustrated by the following simplified excerpt:

Listing 1: Conceptual snippet: creating meta-influencer connectivity to influencers.

```
# Create influencer-of-influencer pattern
meta = create_node(type="meta_influencer", label=1)
for inf in influencer_nodes:
    graph.add_edge(meta, inf)          # meta -> influencer
    for u in followers_of(inf):
        graph.add_edge(inf, u)         # influencer -> regular users
```

4.3 Graph construction and input modalities

All models receive node features $\mathbf{X}$. Only GNNs receive edges E (as sparse adjacency / edge index). Baselines operate on $\mathbf{X}$ alone. This separation is essential for a fair test of "does connectivity help?".

4.4 Phase 2: Real datasets and graph extraction

Phase 2 uses two real graph datasets (as implemented in the project outputs):

- **StackOverflow (machine learning subset)**: users and interactions are turned into a graph; node labels correspond to hierarchical influence proxies used in the project pipeline.
- **WikiVote**: a directed voting graph; again used for node classification under the hierarchical influence framing.

For each dataset, the pipeline constructs features, builds the graph, trains comparable models, and exports plots and summary CSVs for reproducibility.

4.5 Graph visualizer (qualitative verification)

A dedicated interactive visualizer is used to inspect nodes, edges, and multi-hop paths. This is important for two reasons:

1. **Sanity-checking structure**: confirming that synthetic "meta-influencer → influencer" chains exist.
2. **Debugging real graphs**: verifying connectivity, component structure, and that neighborhoods used by message passing reflect intended interactions.

Figure 1 shows a representative visualizer snapshot.

5 Experimental Setup

5.1 Training protocol

Across phases, models are trained for a fixed number of epochs with consistent train/validation/test splits (as defined by the project scripts). We use standard supervised classification losses (cross-entropy or BCE depending on setup) and report:

- **Accuracy**: overall correctness.
- **Macro-F1**: balances class performance when classes are imbalanced.
- **ROC/AUC**: threshold-independent separability.
- **Confusion matrix**: error modes (false positives vs false negatives).
- **Runtime metrics**: average epoch time for training and average inference time.

5.2 Hardware and software

GNN training is implemented in PyTorch / PyTorch Geometric [2]. GPU runs use NVIDIA CUDA (CUDA backend enabled) [5]. The performance analysis focuses on how sparse aggregation workloads map to GPU strengths.

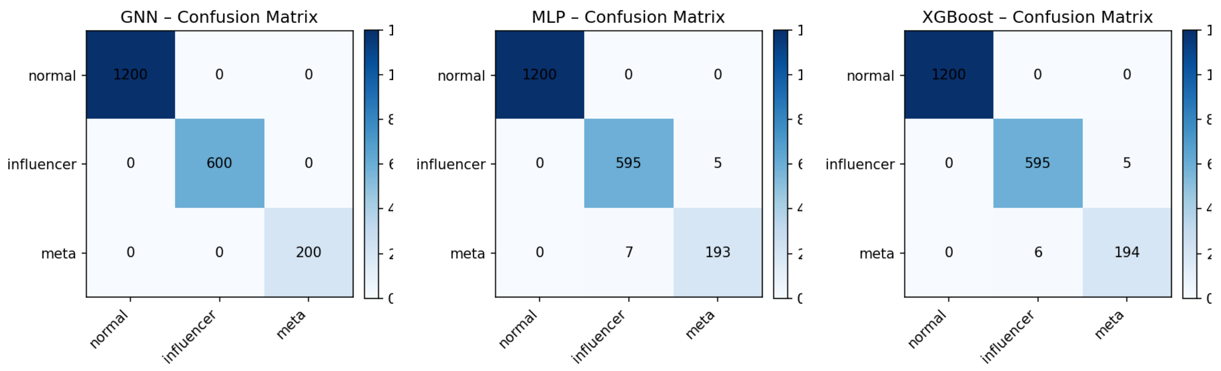
6 Phase 1 Results: Synthetic Benchmark

Phase 1 establishes whether the task genuinely requires structure. We report both multi-class style confusion matrices (separating roles) and binary results (positive = influencer or meta-influencer).

6.1 Multi-class confusion matrices (role separation)

Figures 2a–2c show confusion matrices for GNN, feature-only neural baseline (labeled MLP in plots), and XGBoost on the synthetic dataset. These matrices reveal how each model confuses the classes:

- **GNN**: stronger separation of hierarchical roles due to access to connectivity.
- **Feature-only**: tends to confuse meta-influencers with regular users if their node features overlap.
- **XGBoost**: strong on separable feature regimes, but limited when structure is the primary signal.



(a) GNN (b) Feature-only neural baseline (c) XGBoost

Figure 2: Phase 1 (synthetic) multi-class confusion matrices. These plots highlight which roles are confused when the model has (GNN) vs lacks (baselines) edge information.

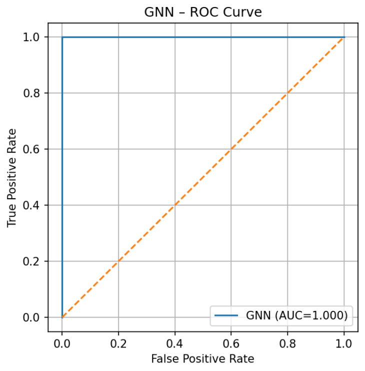
6.2 Binary results: ROC curves, confusion matrices, and learning curves

In the binary setting, the positive class merges influencers and meta-influencers. This tests whether models can at least separate "influential" nodes from regular users. Figures 3 and 4 provide ROC curves and confusion matrices across models. Figures 5 show learning curves for the GNN and the feature-only neural baseline.

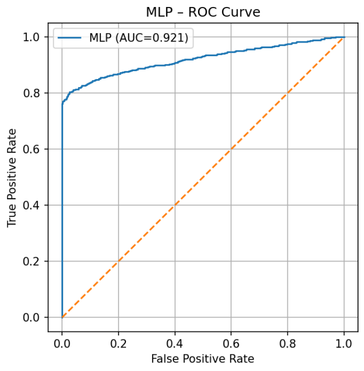
How to read the ROC curve. The ROC curve plots true positive rate vs false positive rate as the classification threshold varies. AUC summarizes separability; higher AUC indicates that the model ranks positive nodes above negative nodes more consistently.

How to read the confusion matrix. The confusion matrix summarizes (TP, FP, TN, FN). For influence detection, false negatives (missing influencers) are often more costly, so we examine recall as well as precision.

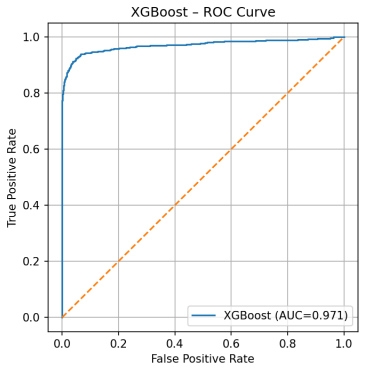
How to read learning curves. Training curves show optimization dynamics. Divergence between training and validation indicates overfitting; slow improvement indicates under-capacity or optimization issues.



(a) GNN ROC

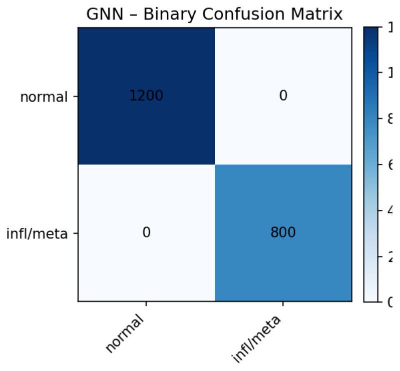


(b) Feature-only ROC

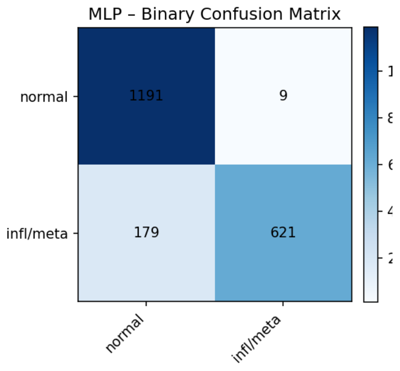


(c) XGBoost ROC

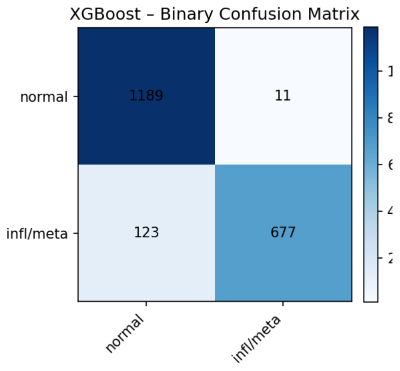
Figure 3: Phase 1 (synthetic) binary ROC curves across models.



(a) GNN CM

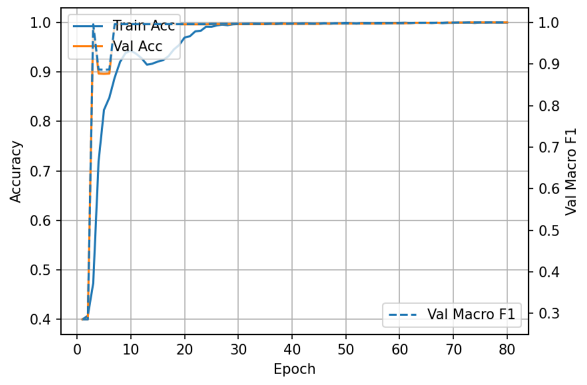


(b) Feature-only CM

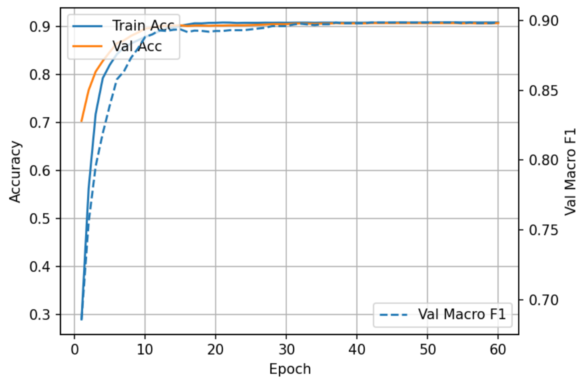


(c) XGBoost CM

Figure 4: Phase 1 (synthetic) binary confusion matrices.



(a) GNN learning curves



(b) Feature-only learning curves

Figure 5: Phase 1 (synthetic) learning curves. These curves help interpret whether performance differences are due to representational limitations (structure vs no structure) or training instability/overfitting.

Table 1: Synthetic phase CPU vs GPU timing and test performance for GNN architectures and layer counts.

arch	layers	avg_train_time_s_cpu	avg_train_time_s_gpu	train_speedup	avg_inference_time_s_cpu	av
gcn	1	0.2375	0.0065	36.6400	0.1121	
gcn	2	0.4715	0.0072	65.5669	0.2251	
gcn	3	0.7179	0.0097	74.1271	0.3400	
gcn	4	0.9483	0.0123	76.9143	0.4575	
graphsage	1	0.0261	0.0276	0.9449	0.0063	
graphsage	2	0.1649	0.0053	31.3674	0.0677	
graphsage	3	0.3015	0.0073	41.4365	0.1340	
graphsage	4	0.4667	0.0081	57.5993	0.1994	

6.3 Phase 1 CPU vs GPU: depth scaling

To study GPU acceleration, we benchmark training and inference times for multiple GNN architectures as model depth (number of layers) increases. Two key effects appear repeatedly in GNN workloads:

- **Overhead at small scale**: for shallow models or small graphs, GPU kernel launch and host-device transfer overhead can dominate, yielding limited speedup.
- **Sparse aggregation acceleration**: as depth increases, neighborhood aggregation becomes heavier; GPUs can parallelize these operations and hide memory latency more effectively.

Table 1 reports measured times and computed speedups. Figures 6 and 7 visualize these trends.

7 Phase 2 Results: Real-World Graphs

Phase 2 evaluates whether conclusions from the synthetic benchmark hold on real data. We report results for StackOverflow (machine learning subset) and WikiVote. For each dataset, we include confusion matrices and ROC curves for GNN, feature-only baseline, and XGBoost, plus training curves for the neural models.

7.1 StackOverflow (Machine Learning subset)

Table 2 summarizes performance metrics. We then analyze plots to interpret where each model succeeds or fails.

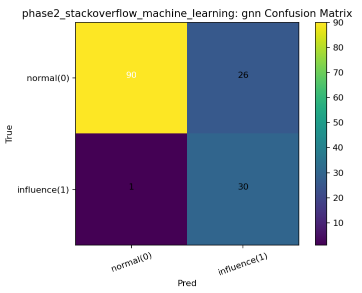
Observations. From the summary metrics and plots:

- **ROC/AUC**: indicates ranking quality; high AUC suggests that the model assigns higher scores to influential users even when a fixed threshold might not perfectly separate classes.
- **Macro-F1**: reflects balanced performance; if the dataset is imbalanced, accuracy alone can be misleading.

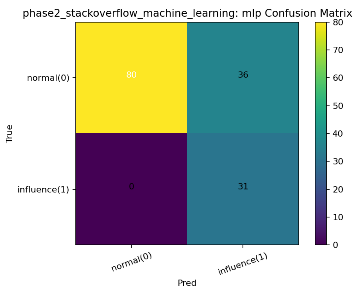
Table 2: Phase 2 results summary on StackOverflow (machine learning subset).

model	accuracy	macro_f1	auc	precision_pos	recall_pos	f1_pos	meta_count	meta_prec	meta_re
xgboost	0.8980	0.8485	0.9714	0.7500	0.7742	0.7619	11	1.0000	0.363
mlp	0.7551	0.7245	0.9095	0.4627	1.0000	0.6327	11	1.0000	1.000
gnn	0.8163	0.7796	0.9011	0.5357	0.9677	0.6897	11	1.0000	1.000

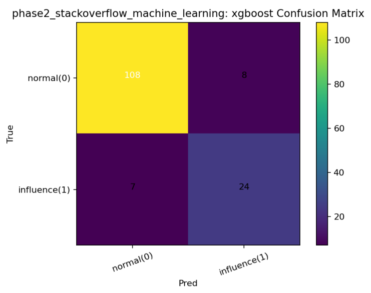
- **Meta-influencer metrics**: the exported summary includes precision/recall/F1 on the "meta" group, which directly targets the influencers-of-influencers phenomenon.



(a) GNN CM

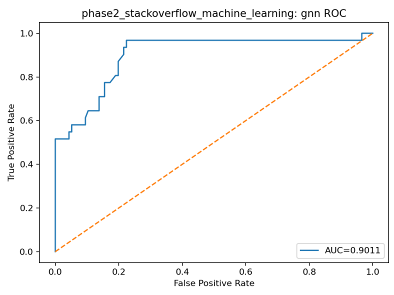


(b) Feature-only CM

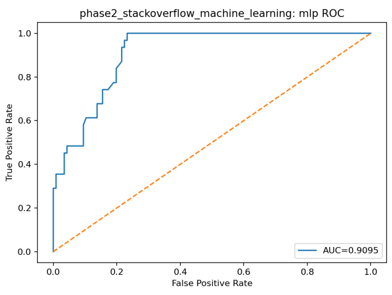


(c) XGBoost CM

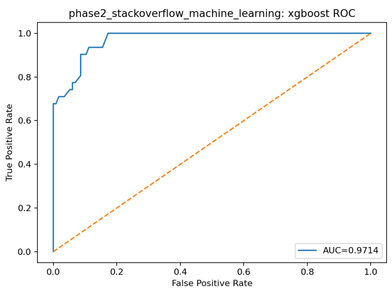
Figure 8: StackOverflow: confusion matrices by model. These reveal whether errors are dominated by false negatives (missing influencers) or false positives (over-predicting influence).



(a) GNN ROC



(b) Feature-only ROC

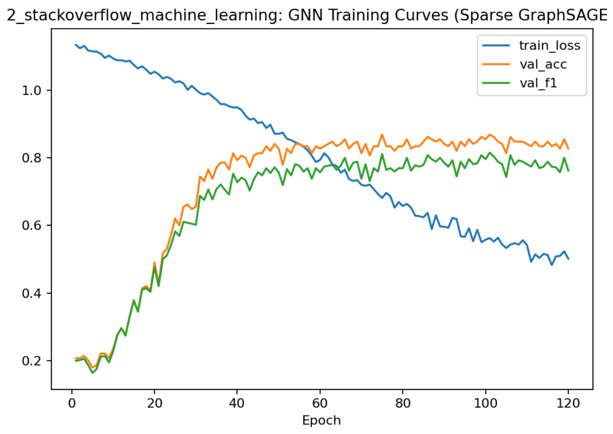


(c) XGBoost ROC

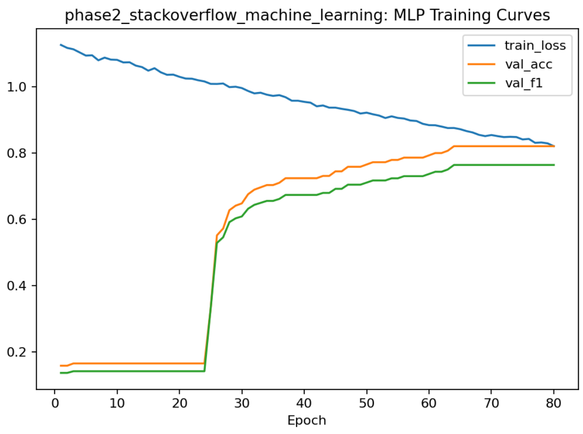
Figure 9: StackOverflow: ROC curves by model. AUC summarizes threshold-independent separability.

Table 3: Phase 2 results summary on WikiVote.

model	accuracy	macro_f1	auc	precision_pos	recall_pos	f1_pos	meta_count	meta_prec	meta_re
xgboost	0.9906	0.9837	0.9994	0.9677	0.9783	0.9730	80	1.0000	0.950
mlp	0.7472	0.6825	0.8774	0.3930	0.8587	0.5392	80	1.0000	0.675
gnn	0.9007	0.8488	0.9415	0.6512	0.9130	0.7602	80	1.0000	0.800



(a) GNN learning curves

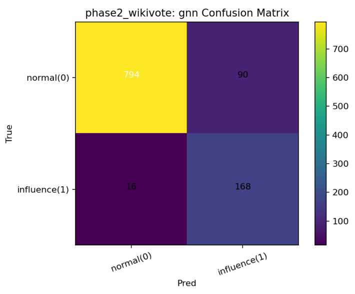


(b) Feature-only learning curves

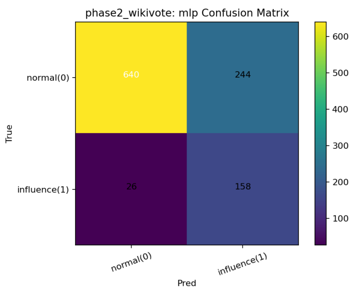
Figure 10: StackOverflow: training dynamics. These curves help diagnose stability and generalization gaps.

7.2 WikiVote

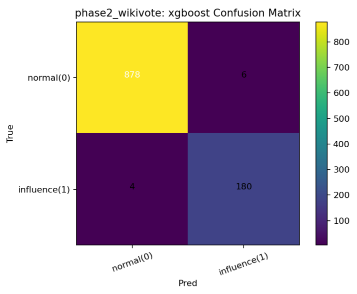
Table 3 summarizes performance metrics on WikiVote. We then interpret plots.



(a) GNN CM

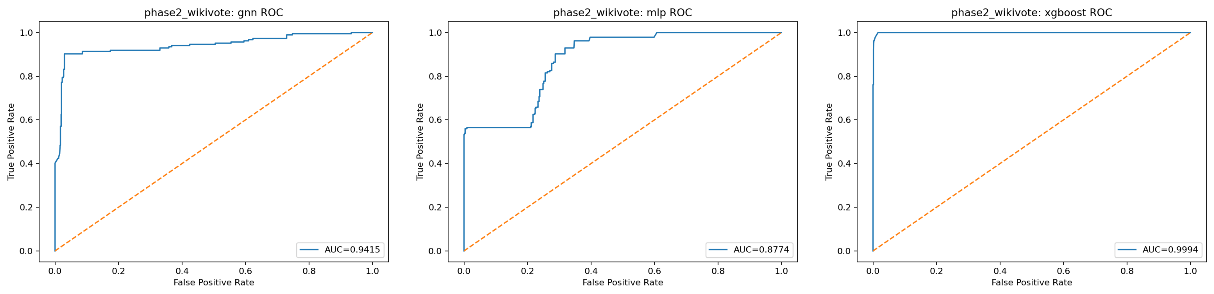


(b) Feature-only CM



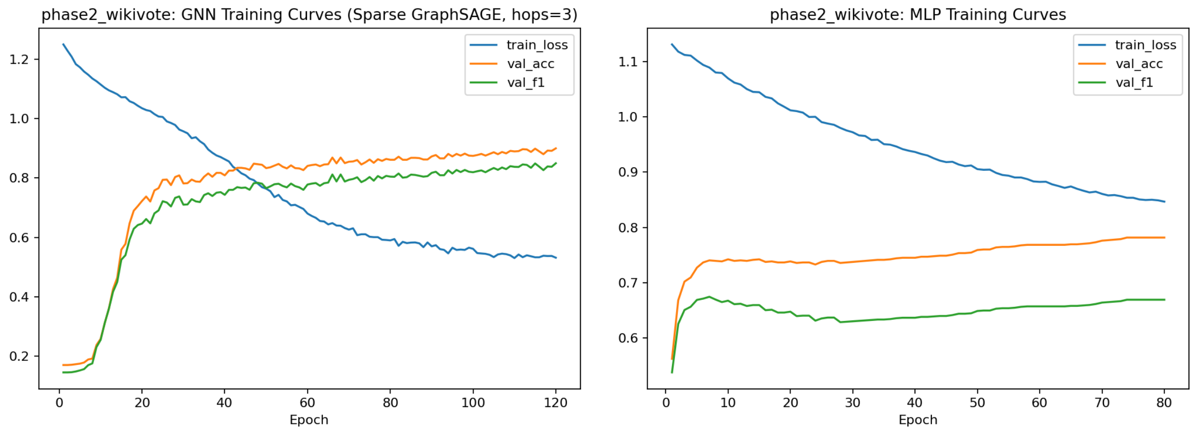
(c) XGBoost CM

Figure 11: WikiVote: confusion matrices by model.



(a) GNN ROC (b) Feature-only ROC (c) XGBoost ROC

Figure 12: WikiVote: ROC curves by model.



(a) GNN learning curves (b) Feature-only learning curves

Figure 13: WikiVote: training dynamics.

8 Phase 2 (Part 2): Hop Sweeps and CPU vs GPU Performance

A critical control knob in GNNs is the *neighborhood depth* (hops). Increasing hops increases the receptive field, potentially improving the ability to detect hierarchical influence—but also increases compute and memory cost due to more message passing.

Table 4 summarizes hop-sweep results across datasets and GNN variants, including CPU vs GPU epoch-time and inference-time speedups.

8.1 Interpreting hop-sweep behavior

We interpret the hop-sweep along two axes:

Predictive performance vs hops. Performance may improve with more hops because meta-influence patterns are multi-hop. However, too many hops can lead to oversmoothing (node representations become similar) or noise aggregation, harming accuracy/F1.

Table 4: Phase 2 (Part 2) CPU vs GPU timing and performance across datasets, GNN variants, and hop counts.

dataset	gnn	hops	avg_epoch_time_s_cpu	avg_epoch_time_s_cuda	train_speedup_cpu_over_gpu
stackoverflow_ml	gcn	1	0.0047	0.0023	2.0634
stackoverflow_ml	gcn	2	0.0081	0.0034	2.4047
stackoverflow_ml	gcn	3	0.0077	0.0042	1.8258
stackoverflow_ml	gcn	4	0.0097	0.0052	1.8576
stackoverflow_ml	sage	1	0.0243	0.0035	6.9928
stackoverflow_ml	sage	2	0.0201	0.0049	4.1054
stackoverflow_ml	sage	3	0.0166	0.0050	3.2972
stackoverflow_ml	sage	4	0.0201	0.0061	3.2667
stackoverflow_ml	sgc	1	0.0041	0.0024	1.7510
stackoverflow_ml	sgc	2	0.0044	0.0025	1.7783
stackoverflow_ml	sgc	3	0.0045	0.0025	1.8164
stackoverflow_ml	sgc	4	0.0049	0.0025	1.9332
wikivote	gcn	1	0.0160	0.0028	5.6721
wikivote	gcn	2	0.0504	0.0043	11.7199
wikivote	gcn	3	0.0614	0.0058	10.5147
wikivote	gcn	4	0.0963	0.0076	12.6231
wikivote	sage	1	0.0179	0.0025	7.0633
wikivote	sage	2	0.0558	0.0045	12.3892
wikivote	sage	3	0.0697	0.0058	12.0810
wikivote	sage	4	0.1065	0.0079	13.4317
wikivote	sgc	1	0.0158	0.0024	6.7318
wikivote	sgc	2	0.0259	0.0024	10.7404
wikivote	sgc	3	0.0298	0.0026	11.6731
wikivote	sgc	4	0.0312	0.0026	11.8132

Runtime vs hops. Runtime generally increases with hops because each layer aggregates over neighbors. GPU speedups often increase with hops because the workload becomes large enough to amortize overhead and exploit parallel sparse aggregation.

9 GPU Acceleration and NVIDIA Architecture: Connecting Results to Hardware

This section explains *why* GPU speedups occur in this project, based on the computation performed by GNN message passing.

9.1 What dominates GNN compute?

Unlike CNNs, where dense matrix multiplications are dominant and Tensor Cores can be central, many GNN workloads are dominated by:

- sparse gather/scatter of neighbor features,
- reduction (sum/mean) over variable-size neighborhoods,
- memory bandwidth and cache behavior (irregular access),
- kernel launch overhead when graphs are small.

9.2 Why NVIDIA GPUs help (and when they do not)

The project's results align with the following mapping:

- **Large speedups at larger hops / deeper models**: more aggregation work increases arithmetic intensity and GPU occupancy.
- **Inference speedups often exceed training speedups**: forward pass is simpler than backward pass and can be more efficiently parallelized.
- **Limited speedup for small graphs or shallow models**: fixed overhead dominates; CPU can be competitive.

9.3 Architecture-relevant GPU features

Observed speedups are most consistent with:

- **Massive thread-level parallelism** (CUDA cores): parallelizing neighbor aggregation across edges.
- **High memory bandwidth**: streaming feature vectors for many edges.
- **Efficient sparse primitives**: scatter/gather and reductions (often implemented via specialized kernels).

10 Discussion: What the Results Mean

10.1 Why GNNs are suited for influencers-of-influencers

The "influencers of influencers" label is inherently relational: a node's importance depends on the importance of nodes it connects to, which is naturally captured by message passing. Feature-only models cannot represent this dependency unless such multi-hop structure is explicitly engineered into features.

10.2 When feature-only models can still win

On some real datasets, feature-only baselines (including XGBoost) can appear strong if features correlate strongly with influence proxies. This does not contradict the value of GNNs; rather it highlights that the benefit of GNNs depends on how much label signal is structural vs purely feature-driven.

10.3 Failure modes and instability

Some configurations in the hop-sweep table show degraded accuracy for certain model/dataset/hop choices. Common causes include:

- oversmoothing for deep message passing,
- class imbalance amplifying threshold sensitivity,
- hyperparameter sensitivity (learning rate, dropout, neighbor sampling, normalization),
- graph sparsity/density differences changing neighborhood statistics.

These are well-known challenges in GNN practice and motivate careful tuning and architecture selection.

11 Limitations

- **Dataset labeling**: hierarchical influence is a complex concept; any proxy label may introduce noise.
- **Hyperparameter breadth**: while multiple architectures and hop settings are explored, exhaustive tuning can further improve performance.
- **Generalization across domains**: performance trends may vary with graph type (directed vs undirected, homophily vs heterophily).

12 Future Work

A promising direction is to build hybrid pipelines:

1. Learn graph-aware embeddings with a GNN.
2. Distill or export these embeddings to feature-only models (MLP/CNN/XGBoost) for fast inference.

This could preserve relational signal while enabling lower-latency production systems. Another direction is to incorporate richer edge types (comments, replies, co-activity) into a heterogeneous GNN and study how different relation channels impact both accuracy and GPU efficiency.

13 Conclusion

This report demonstrates that (i) hierarchical influence detection benefits from relational inductive bias, making GNNs a natural fit, and (ii) NVIDIA GPUs provide substantial acceleration for GNN workloads when the computation is large enough (deeper models or larger hop neighborhoods) to amortize overhead and exploit parallel sparse aggregation. The synthetic-to-real evaluation strategy makes these conclusions robust: Phase 1 verifies the structural necessity, and Phase 2 validates practical relevance and scalability.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[2] Matthias Fey and Jan E. Lenssen. Pytorch geometric: Fast graph representation learning with pytorch. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

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

[5] NVIDIA. Cuda c programming guide, 2025. Accessed 2025-12.

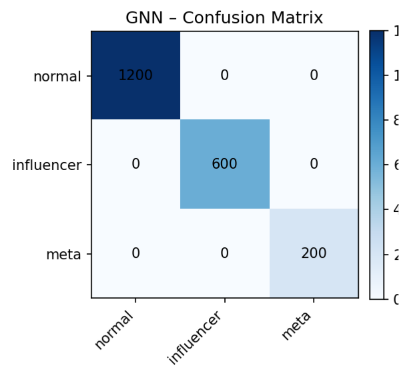
[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

A Appendix: Full Plot Gallery

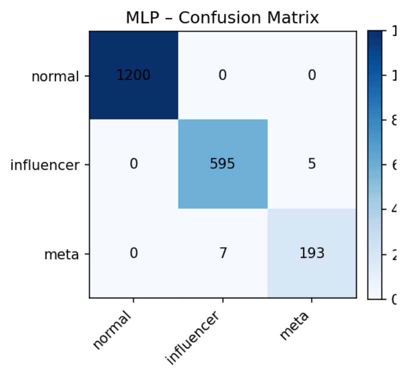
For completeness and reproducibility, we include all exported plots from the project artifacts.

A.1 Phase 1 additional plots

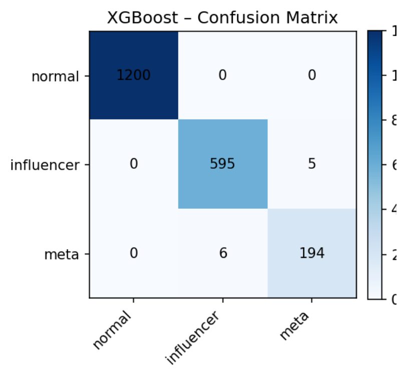
Figures below replicate Phase 1 outputs already discussed, provided for quick reference in a single location.



(a) Synthetic multi-class CM (GNN)

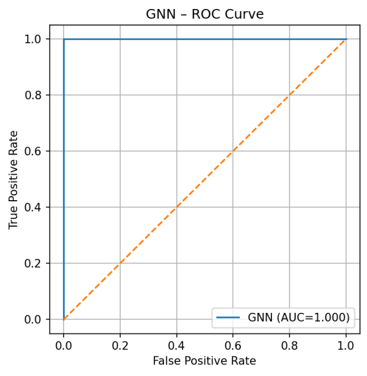


(b) Synthetic multi-class CM (Feature-only)

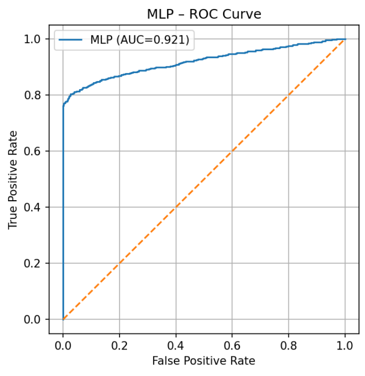


(c) Synthetic multi-class CM (XGBoost)

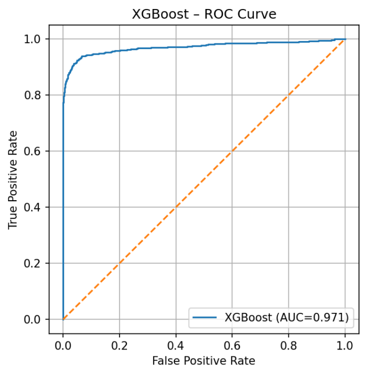
Figure 14: Appendix: Phase 1 synthetic multi-class confusion matrices.



(a) ROC (GNN)

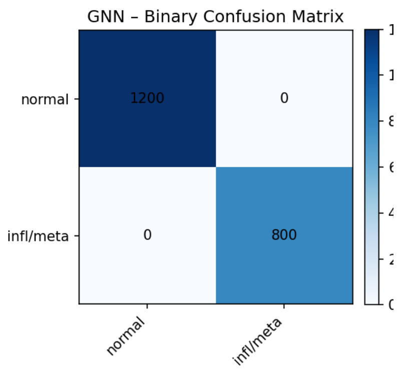


(b) ROC (Feature-only)

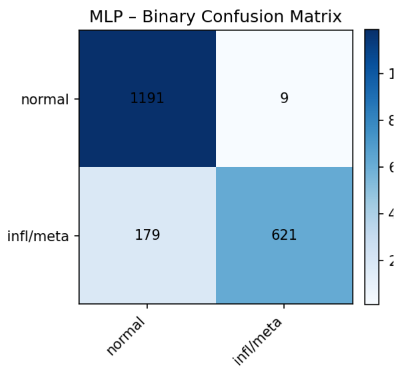


(c) ROC (XGBoost)

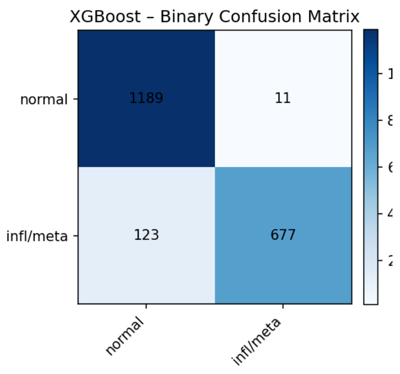
Figure 15: Appendix: Phase 1 synthetic ROC curves.



(a) CM (GNN)

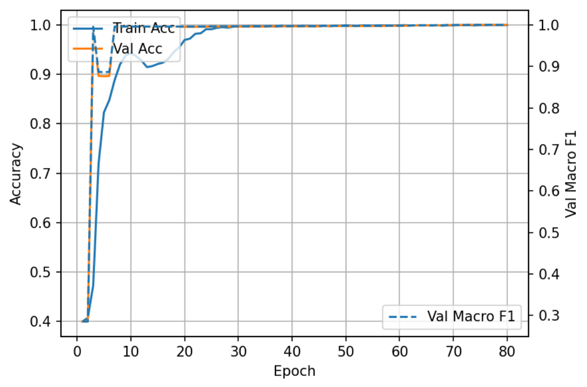


(b) CM (Feature-only)



(c) CM (XGBoost)

Figure 16: Appendix: Phase 1 synthetic binary confusion matrices.


(a) Learning curve (GNN)

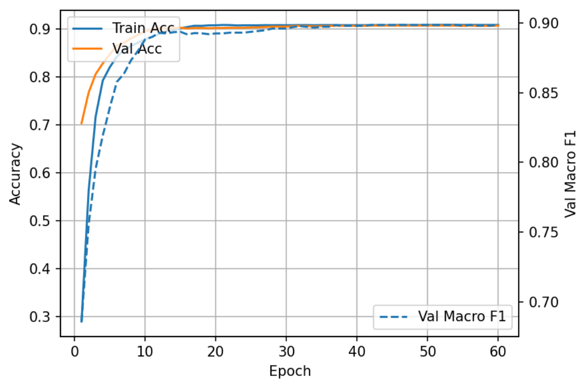
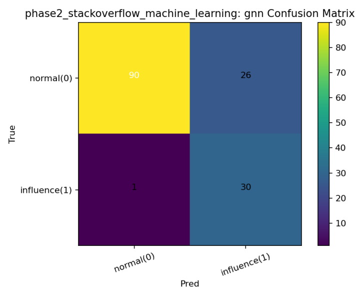
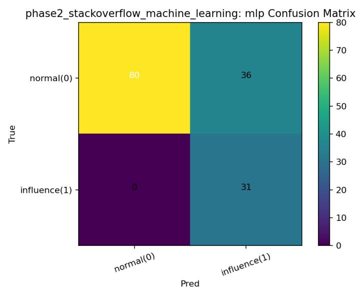

(b) Learning curve (Feature-only)

Figure 17: Appendix: Phase 1 synthetic learning curves.

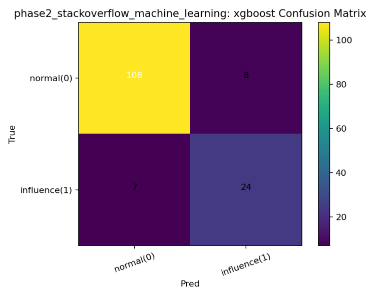
A.2 Phase 2 additional plots



(a) SO CM (GNN)

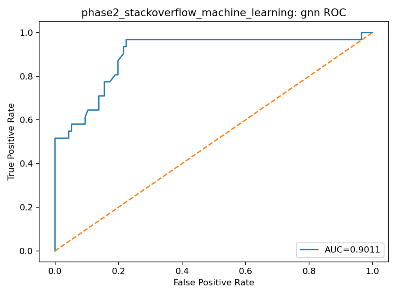


(b) SO CM (Feature-only)

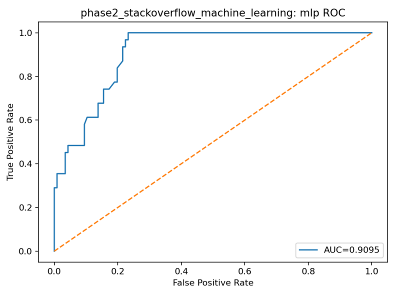


(c) SO CM (XGBoost)

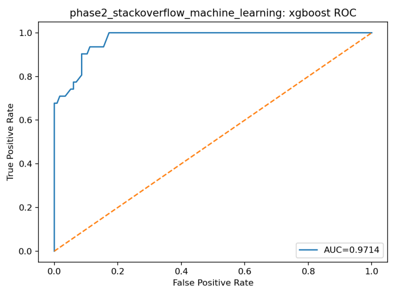
Figure 18: Appendix: StackOverflow confusion matrices.



(a) SO ROC (GNN)

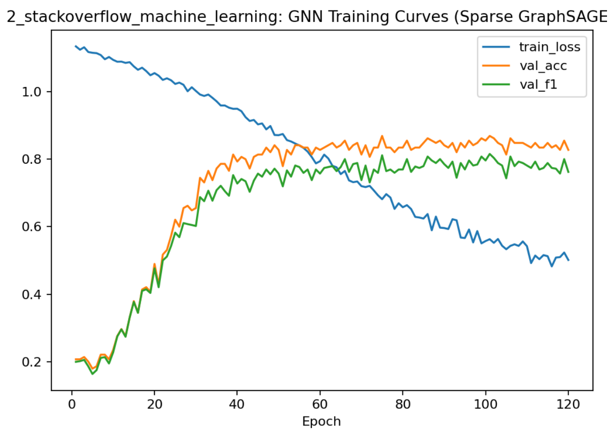


(b) SO ROC (Feature-only)

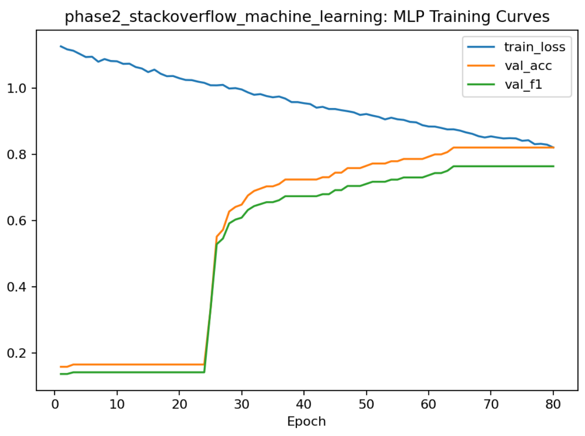


(c) SO ROC (XGBoost)

Figure 19: Appendix: StackOverflow ROC curves.

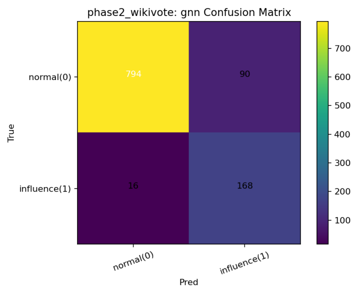


(a) SO learning curves (GNN)

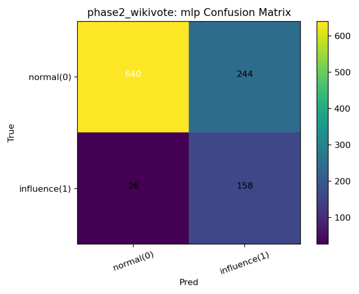


(b) SO learning curves (Feature-only)

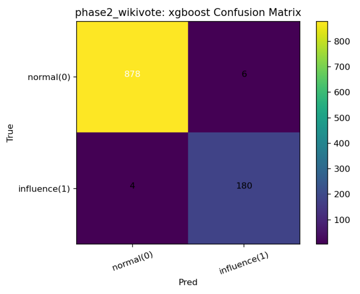
Figure 20: Appendix: StackOverflow learning curves.



(a) WV CM (GNN)

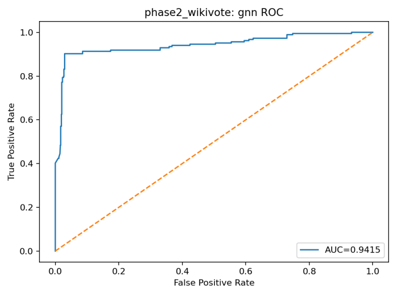


(b) WV CM (Feature-only)

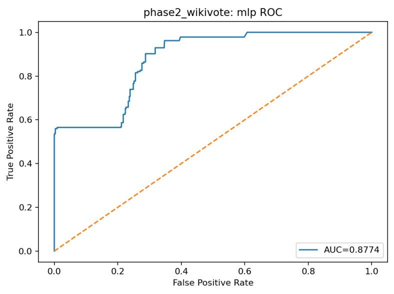


(c) WV CM (XGBoost)

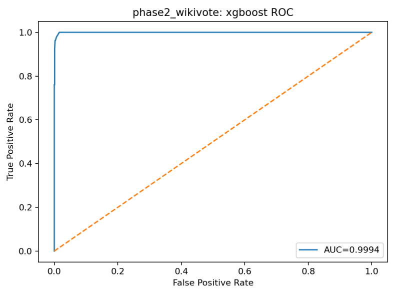
Figure 21: Appendix: WikiVote confusion matrices.



(a) WV ROC (GNN)

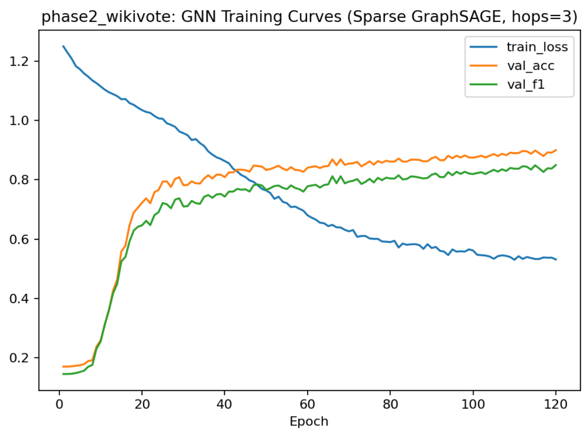


(b) WV ROC (Feature-only)

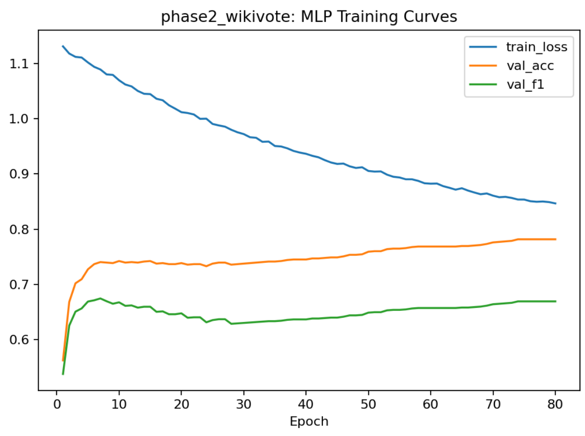


(c) WV ROC (XGBoost)

Figure 22: Appendix: WikiVote ROC curves.



(a) WV learning curves (GNN)



(b) WV learning curves (Feature-only)

Figure 23: Appendix: WikiVote learning curves.

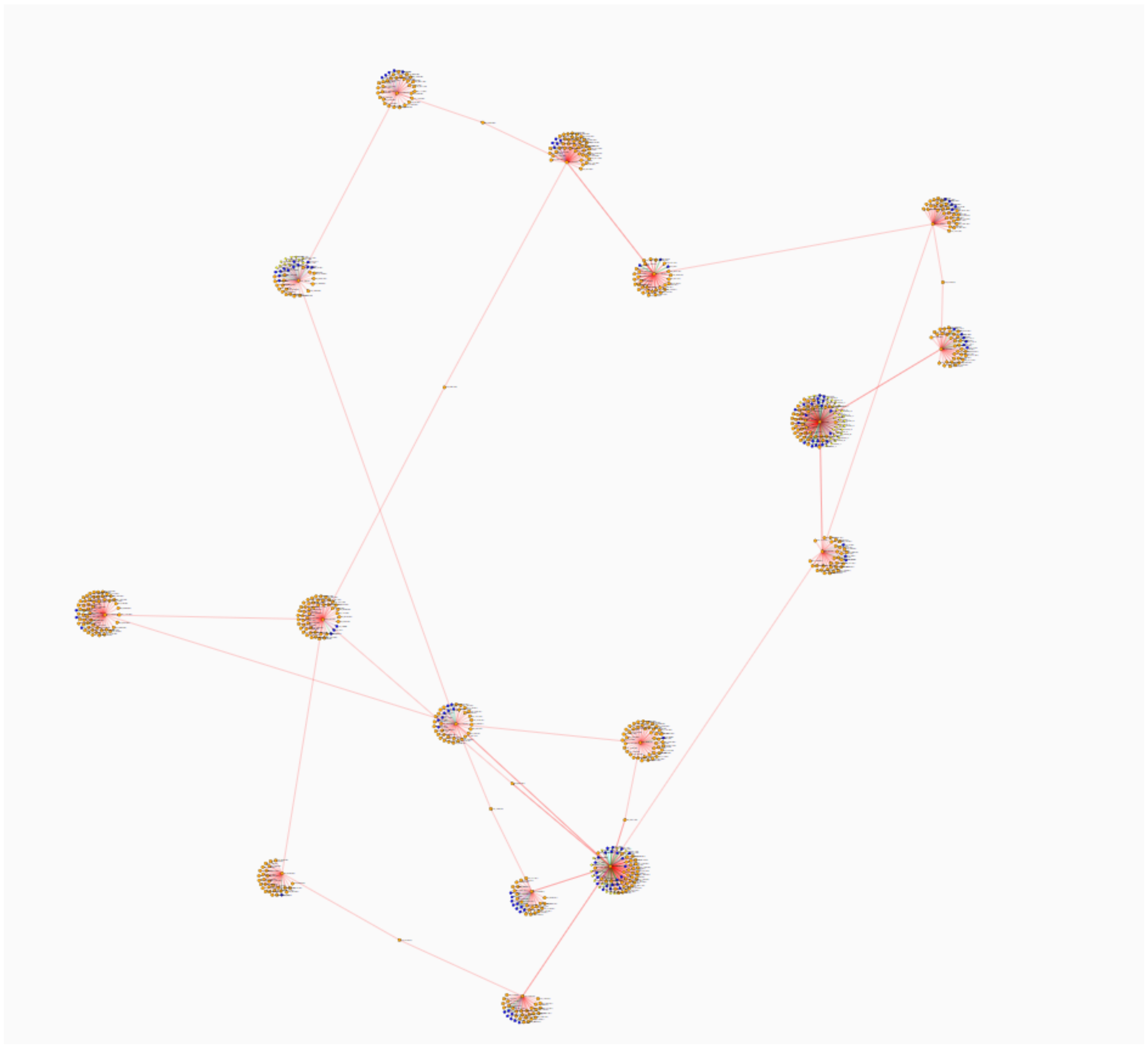


Figure 1: Graph visualizer snapshot showing clusters and connecting edges. The visual structure helps confirm the presence of multi-hop influence patterns (meta-influencers connected to influencers who connect to many regular users).

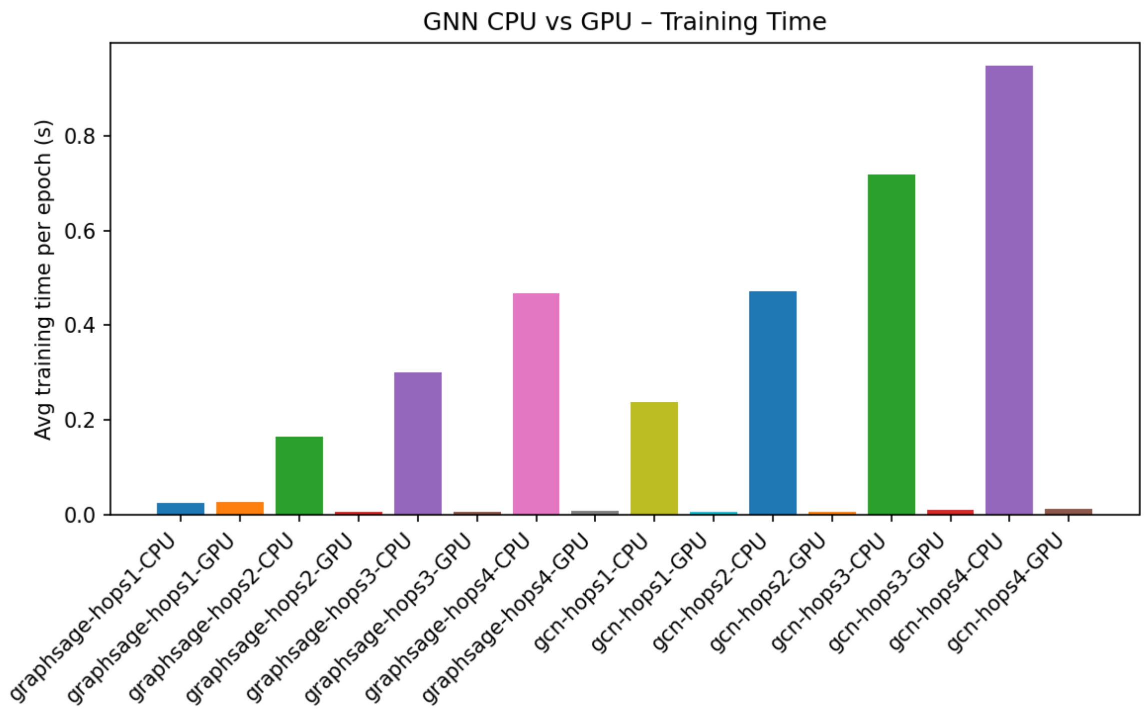


Figure 6: Phase 1 synthetic: average training time per epoch for CPU vs GPU across GNN variants and depths. The GPU advantage increases as depth increases and aggregation dominates.

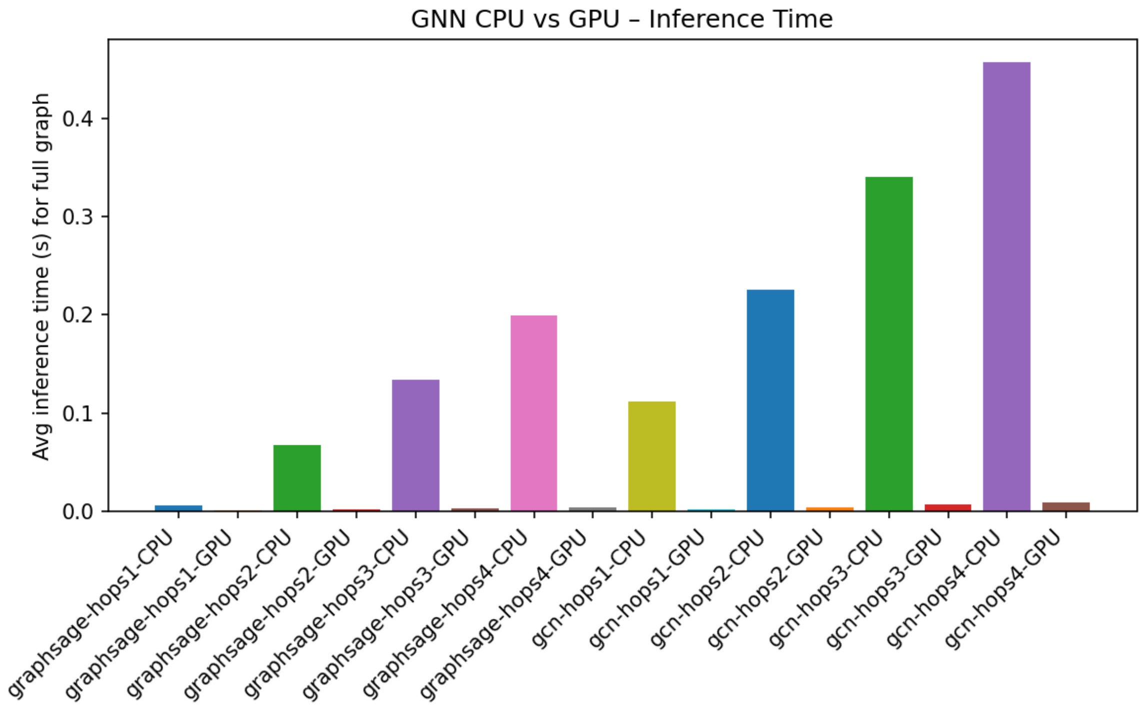


Figure 7: Phase 1 synthetic: average inference time for CPU vs GPU across GNN variants and depths. Inference speedups are often larger because forward pass is more regular than backprop and benefits from parallel sparse ops.