Exploring Graph Neural Networks: Design, Performance, Comparative Evaluation

Can NVIDIA GPUs Help?

Ashik Ali Shaik

George Mason University

December 14, 2025

Abstract (Problem, Approach)

Problem: Detect *influencers-of-influencers* (meta-influencers), where the label depends on **multi-hop relational structure**, not only node-level features.

Approach:

- Phase 1 (Synthetic): engineer a controlled graph with (meta-influencer, influencer, normal) structure; validate that **GNNs exploit topology**.
- Phase 2 (Real): build graphs from real datasets (e.g., WikiVote, StackOverflow ML) and compare **GNN vs MLP/CNN vs XGBoost**.
- Systems Study: benchmark **CPU vs NVIDIA GPU** training/inference to understand which compute patterns benefit.

Abstract (Key Findings)

- **GNNs** improve detection when the class definition is **structural** (multi-hop influence), while feature-only baselines can miss hierarchical influence patterns.
- **GPU benefits are scale-dependent:** for small graphs overhead can dominate; for larger graphs/hops, GPU acceleration becomes meaningful.
- The observed speedups align with GPU strengths in **parallel sparse aggregation** (message passing) and **memory bandwidth** (neighbor reads).

Motivation: Why "Influencers of Influencers" is a Graph Problem

- In this task, the label is not determined by raw features alone (e.g., follower count).
- A **meta-influencer** may have few direct followers, but their followers influence many others (a **multi-hop** effect).
- Therefore, the key signal lives in **connectivity patterns** across hops.

Research Questions

1. When the label is structural, do **GNNs** outperform feature-only models (MLP/CNN/XGBoost)?
2. How do different GNN architectures (GCN, GraphSAGE, GAT/SGC variants) compare?
3. Can **NVIDIA GPUs** accelerate GNN training and inference? When and why?
4. Which GPU architecture aspects matter most for GNN workloads?

Contributions

- **Designed** a synthetic influence dataset with explicit meta-influencer hierarchy.
- **Implemented** end-to-end pipelines: data $\rightarrow$ graph $\rightarrow$ training $\rightarrow$ evaluation.
- **Compared** multiple GNNs against baselines (MLP/CNN-style, XGBoost).
- **Measured** CPU vs GPU training and inference across model depth / hops.
- **Analyzed** observed speedups in terms of GPU compute patterns (sparse aggregation, bandwidth).

GNNs vs MLP/CNN (Intuition)

- **MLP/XGBoost:** input is a feature vector per node; assumes i.i.d. samples.
- **CNN:** learns local patterns but assumes grid structure (images, sequences).
- **GNN:** learns from **features + edges**; information propagates via neighbors.

Message Passing (Core Mechanism)

$$h_v^{(k)} = \sigma\Big(W^{(k)} \cdot \mathrm{AGG}\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\Big)$$

- Each layer aggregates neighbor embeddings (often sparse, irregular memory access).
- Deeper layers (or higher hops) capture longer-range influence.

GNNs Explored in This Project

- **GCN:** normalized neighbor sum; strong baseline.
- **GraphSAGE:** sampling/aggregation; scalable.
- **SGC / simplified variants:** linearized propagation (fewer non-linearities).

We evaluate across **depth (layers)** and/or **hops** to measure structural information gain & compute cost.

End-to-End Project Flow (Paper-Style Overview)

1. Define task: detect influencers-of-influencers (meta-influencers).
2. Phase 1: **Synthetic** graph generation (controlled topology).
3. Train and evaluate: **GNN vs baselines** on synthetic.
4. Phase 1 Part 2: **CPU vs GPU** timing on synthetic.
5. Phase 2: **Real** datasets $\rightarrow$ graphs (WikiVote, StackOverflow ML).
6. Train and evaluate: accuracy/F1/AUC, confusion matrices, training curves.
7. Phase 2 Part 2: **CPU vs GPU** timing vs hops; analyze scaling.

Models & Baselines

Category	Models
GNN	GCN, GraphSAGE, (SGC variants)
Baselines	MLP/CNN-style feature model, XGBoost

Inputs to Each Model (Fair Comparison)

Model	Node Features	Graph Relations (Edges/Hops)
XGBoost	✓	×
MLP/CNN-style	✓	×
GNN	✓	✓

Key idea: all models see the same node-level features; only GNNs can exploit structure.

Evaluation Metrics

- Classification: Accuracy, Macro-F1, AUC; Positive-class Precision/Recall/F1
- Diagnostics: Confusion Matrix, ROC curves, Training curves
- Systems: Average training time per epoch; inference latency; CPU vs GPU speedup

Phase 1 (Part 1): Synthetic Dataset Design

- Constructed a controlled social graph with three groups:
  - **Meta-influencers** (influencers of influencers) label=1
  - **Influencers** label=1
  - **Normal users** label=0
- Design goal: make meta-influencers **not trivially separable** by degree alone.
- Node features provided to all models (tabular); edges provided only to GNNs.

Synthetic Data: Tabular View (Used by Baselines)

We also export a tabular CSV for feature-only baselines (XGBoost/MLP/CNN-style).

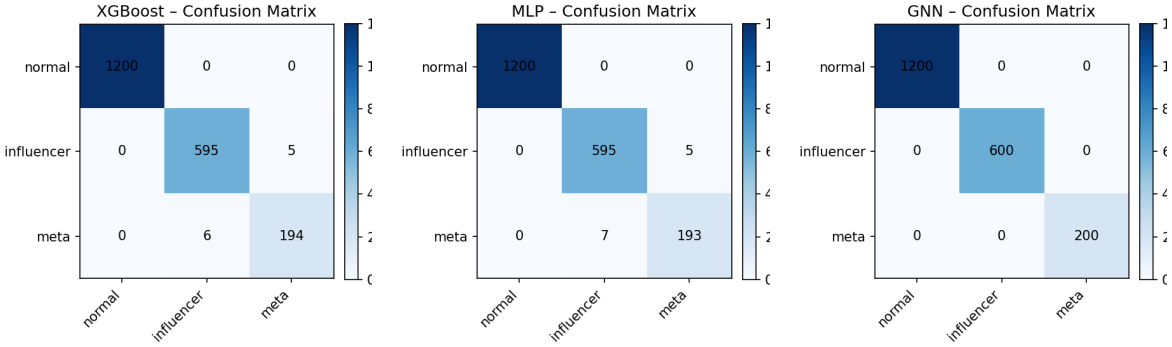
```
# Example: tabular features for baseline models
# columns like: follower_count, following_count, post_count, ...
df = pd.read_csv("phase1_synthetic_influence_tabular.csv")
X = df[feature_cols].values
y = df["label"].values
```

Synthetic Data: Graph Object (Used by GNN)

```
# Example: PyG-style graph object for GNN
data = torch.load("phase1_hetero_gnn_data.pt")
# data.x : node features
# data.edge_index : graph connectivity
# data.y : labels
```

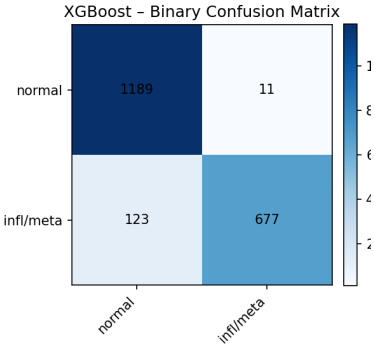
Why it matters: the same features can be ambiguous; edges disambiguate via multi-hop propagation.

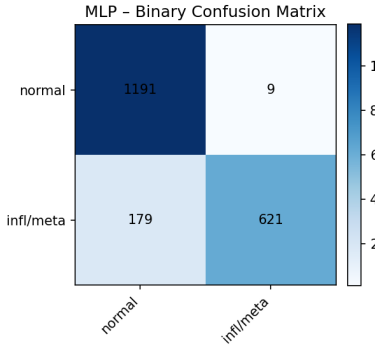
Phase 1 Results: Confusion Matrices (Multiclass / 3-type)

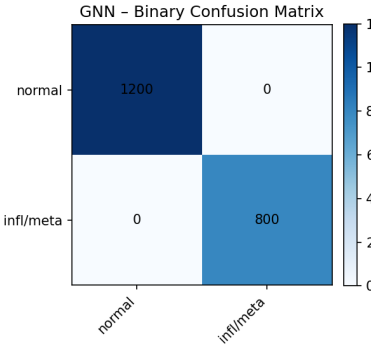


Left to right: XGBoost, MLP/CNN-style, GNN. This highlights where structure-aware learning improves class separation.

Phase 1 Results: Binary Task (Meta/Influencer vs Others)

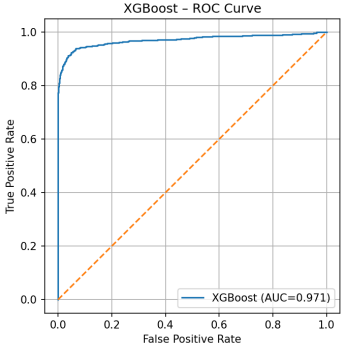


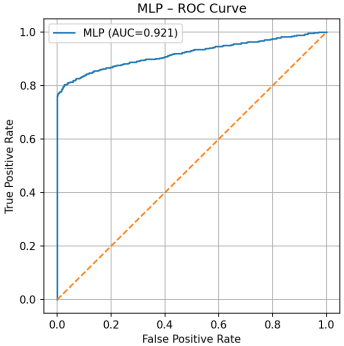


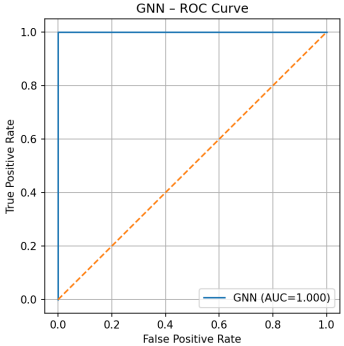


Binary formulation isolates the key detection objective: identify (influencer, meta-influencer) vs normal.

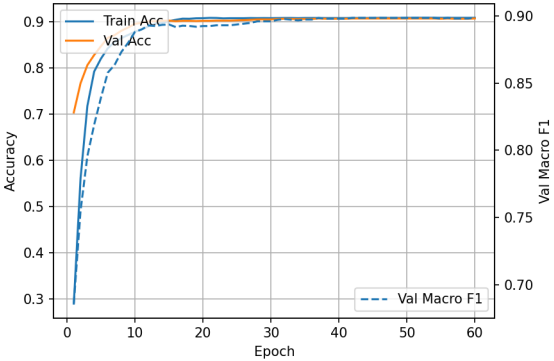
Phase 1 Results: ROC Curves (Binary)

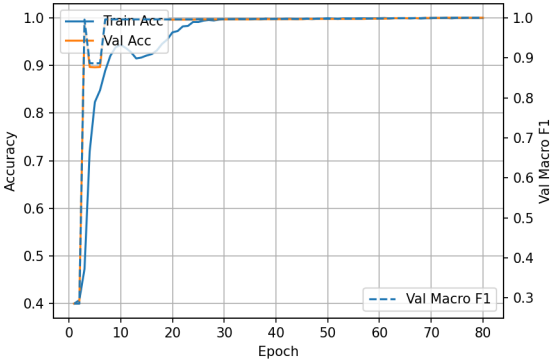






Phase 1 Results: Training Curves





Training curves help validate convergence behavior and stability.

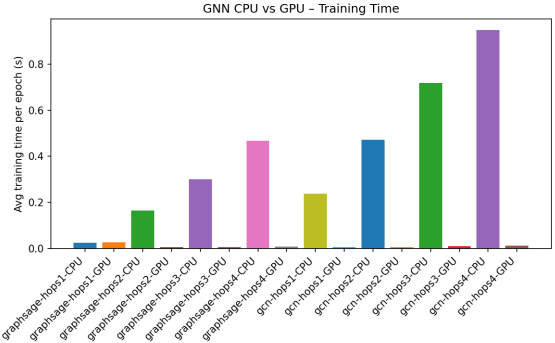
Phase 1 (Part 2): CPU vs GPU Benchmark (Summary Table)

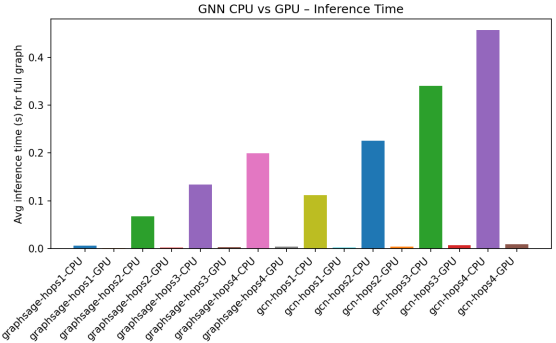
arch	layers	train_{cpu} s	train_{gpu} s	$\text{train}_{speedup}$	infer_{cpu} s	infer_{gpu} s	$\text{infer}_{speedup}$	acc_{cpu}	acc_{gpu}	f1_{cpu}	f1_{gpu}
gcn	1	0.2375	0.0065	36.6400	0.1121	0.0023	48.1739	0.9995	0.9980	0.9995	0.9979
gcn	2	0.4715	0.0072	65.5669	0.2251	0.0044	50.6140	0.9980	0.9980	0.9979	0.9979
gcn	3	0.7179	0.0097	74.1271	0.3400	0.0067	50.8422	0.9930	0.9950	0.9927	0.9948
gcn	4	0.9483	0.0123	76.9143	0.4575	0.0091	50.5274	0.9815	0.6955	0.9808	0.6921
graphsage	1	0.0261	0.0276	0.9449	0.0063	0.0014	4.3954	0.9985	0.9970	0.9984	0.9969
graphsage	2	0.1649	0.0053	31.3674	0.0677	0.0023	29.8405	0.9965	0.9965	0.9964	0.9964
graphsage	3	0.3015	0.0073	41.4365	0.1340	0.0032	42.1125	0.9990	0.9965	0.9990	0.9964
graphsage	4	0.4667	0.0081	57.5993	0.1994	0.0044	44.9811	0.9965	1.0000	0.9964	1.0000

Interpretation:

- Some settings show very large GPU speedups (parallel aggregation).
- Some small/shallower settings show limited speedup or even slowdown (overhead).

Phase 1 (Part 2): Timing Plots





GPU gains depend on depth/compute: deeper propagation increases parallel work, favoring GPU.

Phase 2 Datasets

We evaluate on real datasets to validate whether synthetic findings hold at scale:

- **WikiVote:** voting relations $\rightarrow$ graph structure.
- **StackOverflow (ML subset):** interactions $\rightarrow$ graph structure.

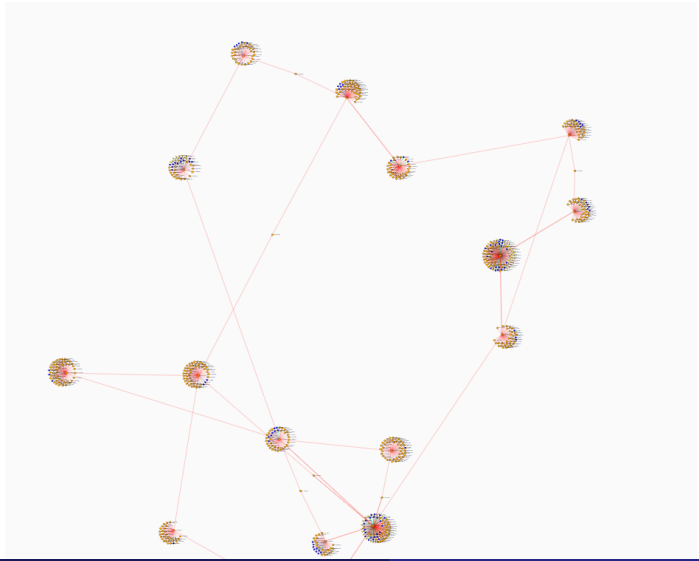
Graph Visualizer (Debugging & Validation)

To validate graph construction, I built an interactive visualizer:

- Export graphs to JSON: `graph_for_visualizer_small.json`, `graph_for_visualizer_medium.json`
- Visualize in browser: `graph_visualizer.html`
- Ensures the intended meta-influence paths exist before training.

```
# Export for visualizer
# nodes: id, type, features; edges: (src, dst, relation)
python export_graph_for_visualizer_small.py
```

Graph Visualizer Demo (Live)



Demo

- Open the visualizer HTML in a browser.
- Load `vis/graph_for_visualizer_small.`
- Enter a user id + max hops; expand neighbors by clicking nodes.
- Use zoom/pan to trace multi-hop influence paths.

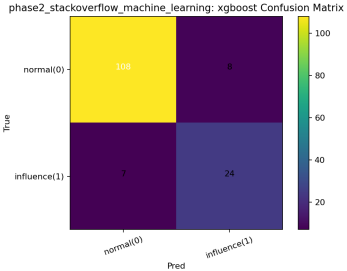
Open (local):
`vis/graph_visualizer.html`

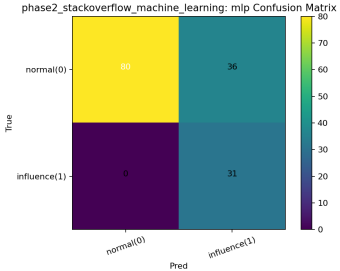
Phase 2 (Part 1): Results Summary — StackOverflow ML

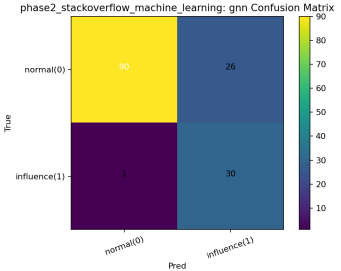
model	accuracy	macro$_f$1	auc	precision$_p$os	recall$_p$os	f1$_p$os	meta$_c$ount	meta$_p$rec	meta$_r$ec	meta$_f$1
xgboost	0.898	0.848	0.971	0.750	0.774	0.762	11	1.000	0.364	0.533
mlp	0.755	0.724	0.909	0.463	1.000	0.633	11	1.000	1.000	1.000
gnn	0.816	0.780	0.901	0.536	0.968	0.690	11	1.000	1.000	1.000

Notes: Table includes macro-F1 and meta-class metrics to evaluate influencer-of-influencer detection.

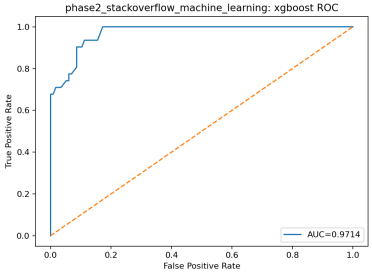
Phase 2 (Part 1): StackOverflow ML — Confusion Matrices

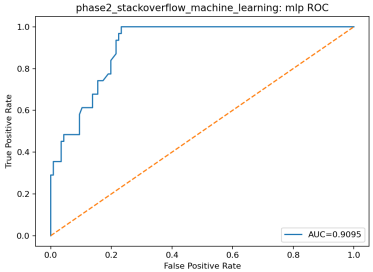


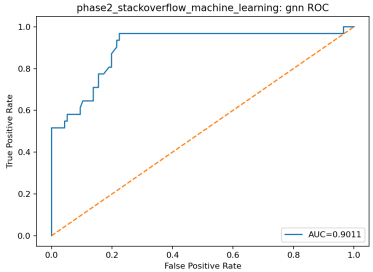




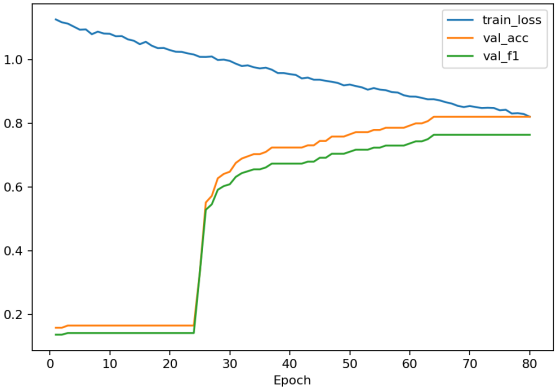
Phase 2 (Part 1): StackOverflow ML — ROC Curves

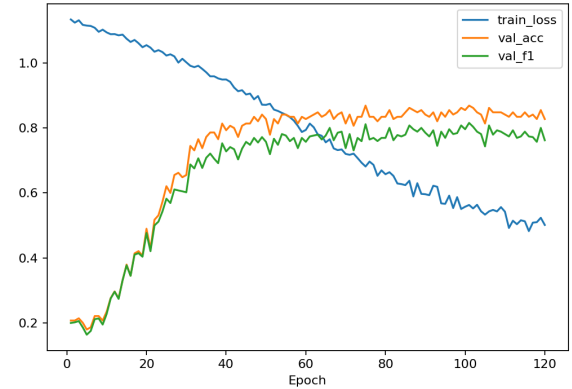






Phase 2 (Part 1): StackOverflow ML — Training Curves

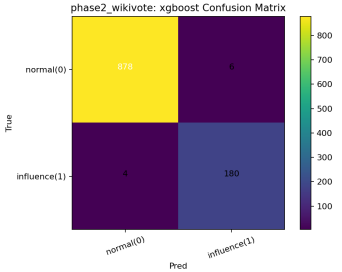


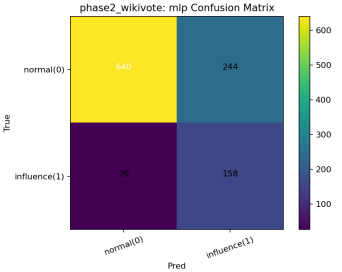


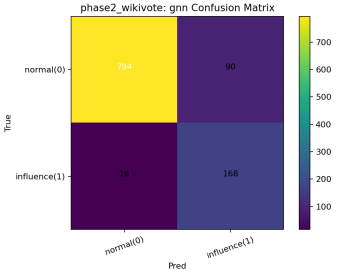
Phase 2 (Part 1): Results Summary — WikiVote

model	accuracy	macro$_f$1	auc	precision$_p$os	recall$_p$os	f1$_p$os	meta$_c$ount	meta$_p$rec	meta$_r$ec	meta$_f$1
xgboost	0.991	0.984	0.999	0.968	0.978	0.973	80	1.000	0.950	0.974
mlp	0.747	0.683	0.877	0.393	0.859	0.539	80	1.000	0.675	0.806
gnn	0.901	0.849	0.941	0.651	0.913	0.760	80	1.000	0.800	0.889

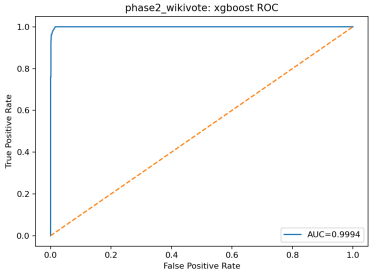
Phase 2 (Part 1): WikiVote — Confusion Matrices

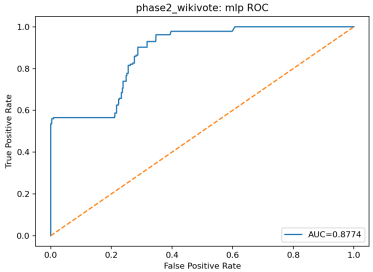


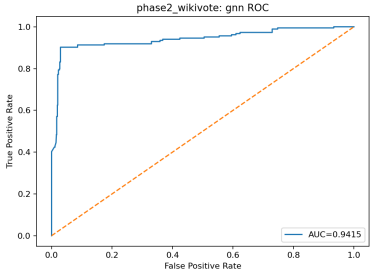




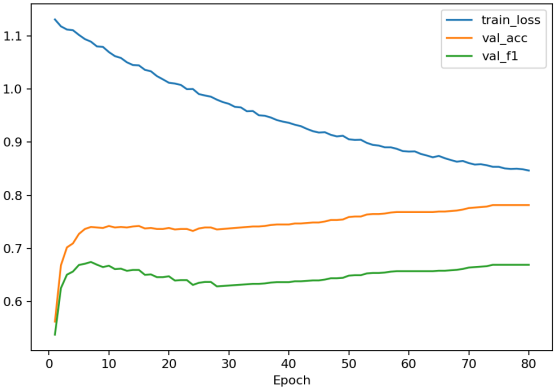
Phase 2 (Part 1): WikiVote — ROC Curves

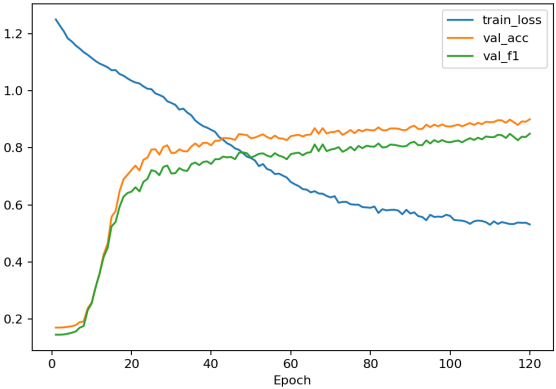






Phase 2 (Part 1): WikiVote — Training Curves

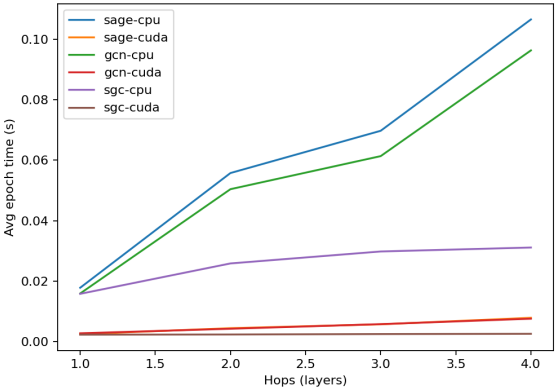


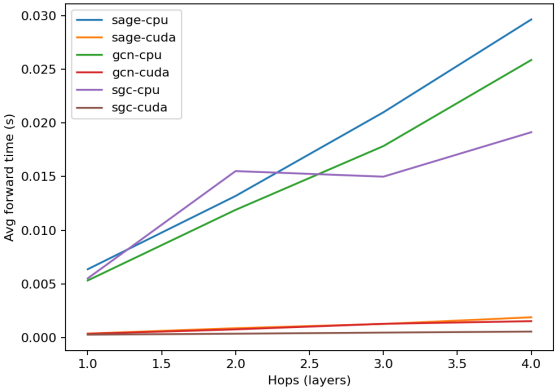


Why GNNs Win for Meta-Influence Detection

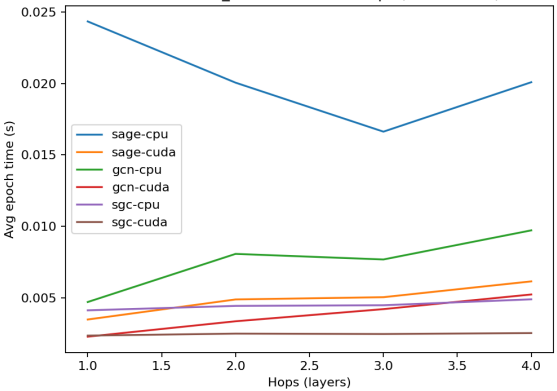
- Meta-influence is defined by **multi-hop** patterns (neighbors-of-neighbors).
- Feature-only models collapse these patterns into a single vector, losing relational signal.
- GNNs explicitly propagate information through the graph, enabling:
  - community-aware embeddings,
  - hop-aware influence diffusion,
  - better separation of structurally-defined classes.

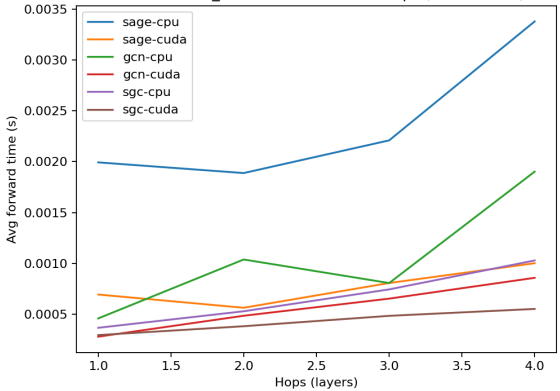
Phase 2 (Part 2): CPU vs GPU Timing vs Hops (Plots)





Phase 2 (Part 2): CPU vs GPU Timing vs Hops (StackOverflow ML)





Phase 2 (Part 2): Speedup Table — WikiVote (All Hops)

dataset	gnn	hops	avg$_{e}$poch$_{t}$ime$_{c}$pu	avg$_{e}$poch$_{t}$ime$_{c}$uda	train$_{s}$pdup$_{c}$pu$_{o}$ver$_{g}$pu	avg$_{i}$nfer$_{t}$ime$_{s}$c pu	avg$_{i}$nfer$_{t}$ime$_{s}$c uda	infer$_{s}$pdup$_{c}$pu$_{o}$
wikivote	gcn	1	0.0160	0.0028	5.6721	0.0053	0.0004	15.1846
wikivote	gcn	2	0.0504	0.0043	11.7199	0.0119	0.0008	15.5672
wikivote	gcn	3	0.0614	0.0058	10.5147	0.0178	0.0013	13.9058
wikivote	gcn	4	0.0963	0.0076	12.6231	0.0259	0.0015	16.8932
wikivote	sage	1	0.0179	0.0025	7.0633	0.0064	0.0004	16.9650
wikivote	sage	2	0.0558	0.0045	12.3892	0.0132	0.0009	15.0735
wikivote	sage	3	0.0697	0.0058	12.0810	0.0210	0.0013	16.5031
wikivote	sage	4	0.1065	0.0079	13.4317	0.0296	0.0019	15.6630
wikivote	sgc	1	0.0158	0.0024	6.7318	0.0055	0.0003	20.2550
wikivote	sgc	2	0.0259	0.0024	10.7404	0.0155	0.0004	42.8834
wikivote	sgc	3	0.0298	0.0026	11.6731	0.0150	0.0005	32.2170
wikivote	sgc	4	0.0312	0.0026	11.8132	0.0191	0.0006	34.1484

Phase 2 (Part 2): Speedup Table — StackOverflow ML (All Hops)

dataset	gnn	hops	$\text{avg}_{epoch}{}_{time}{}_{cpu}$	$\text{avg}_{epoch}{}_{time}{}_{cuda}$	$\text{train}_{speedup}{}_{cpu}{}_{over}{}_{gpu}$	$\text{avg}_{infer}{}_{time}{}_{cpu}$	$\text{avg}_{infer}{}_{time}{}_{cuda}$	$\text{infer}_{speedup}$
stackoverflow_ml	gcn	1	0.0047	0.0023	2.0634	0.0005	0.0003	1.6380
stackoverflow_ml	gcn	2	0.0081	0.0034	2.4047	0.0010	0.0005	2.1362
stackoverflow_ml	gcn	3	0.0077	0.0042	1.8258	0.0008	0.0007	1.2339
stackoverflow_ml	gcn	4	0.0097	0.0052	1.8576	0.0019	0.0009	2.2142
stackoverflow_ml	sage	1	0.0243	0.0035	6.9928	0.0020	0.0007	2.8685
stackoverflow_ml	sage	2	0.0201	0.0049	4.1054	0.0019	0.0006	3.3428
stackoverflow_ml	sage	3	0.0166	0.0050	3.2972	0.0022	0.0008	2.7337
stackoverflow_ml	sage	4	0.0201	0.0061	3.2667	0.0034	0.0010	3.3695
stackoverflow_ml	sgc	1	0.0041	0.0024	1.7510	0.0004	0.0003	1.2414
stackoverflow_ml	sgc	2	0.0044	0.0025	1.7783	0.0005	0.0004	1.3834
stackoverflow_ml	sgc	3	0.0045	0.0025	1.8164	0.0007	0.0005	1.5350
stackoverflow_ml	sgc	4	0.0049	0.0025	1.9332	0.0010	0.0006	1.8603

What the Results Imply About NVIDIA GPU Benefits

Observed acceleration aligns with GPU strengths in **parallel sparse aggregation**:

GNN Compute Pattern	**NVIDIA GPU Feature**
Sparse neighbor aggregation	Massive parallel CUDA cores
Irregular neighbor reads	High memory bandwidth, cache hierarchy
Batching many nodes/edges	High throughput kernels
Multi-hop propagation	More parallel work per iteration (amortizes overhead)

Key message: speedup is driven more by sparse parallelism + bandwidth than dense GEMMs.

When GPU Helps vs When It Doesn't (From Experiments)

- **Small graphs / shallow models:** CPU can be competitive due to GPU kernel launch & transfer overhead.
- **Larger graphs / more hops:** GPU speedups increase as message passing dominates compute.
- **Model differences:** architectures with heavier aggregation/propagation benefit more.

Overall Results: What We Learned

- **Accuracy/Quality:** structure-aware models (GNNs) improve detection when labels depend on multi-hop relations.
- **Compute:** GPUs provide meaningful gains when graph workloads are large enough to saturate parallelism.
- **Design insight:** for influencer-of-influencer detection, the graph representation is the decisive factor.

Conclusions

1. GNNs are best suited for detecting hierarchical influence because they model relational structure directly.
2. Feature-only baselines can succeed when structural patterns correlate strongly with features, but they can fail when structure is the label definition.
3. NVIDIA GPUs accelerate GNN workloads primarily through parallel sparse aggregation and bandwidth, with speedups increasing with hops/scale.

Future Scope (Research Direction)

Hybrid approach: encode relations into features so MLP/CNN/XGBoost can benefit:

- Compute graph embeddings (Node2Vec / GNN embeddings / centrality features).
- Train fast feature-only models on these enriched embeddings for low-latency inference.
- Explore deployment tradeoffs: GNN training offline + lightweight inference online.

Thank You

Questions?

Appendix: Key Code Entry Points

- Synthetic + baselines: `full_exp.py`, `full_final.py`, `ful_exp_binary.py`
- GPU benchmarking: `gpu_gnn.py`
- Phase 2 real data: `part1_phase2.py`, `part2_phase2.py`
- Visualizer exports: `export_graph_for_visualizer_*.py`

Appendix: More Notes for Defense Q&A

- **Fairness:** same node features across models; only GNN sees edges.
- **Why macro-F1:** handles class imbalance; meta-influencers are rare.
- **Why hops/layers:** approximates how far influence propagates.
- **Performance bottlenecks:** sparse aggregation and memory access dominate.