Reddit
Social Network Analysis

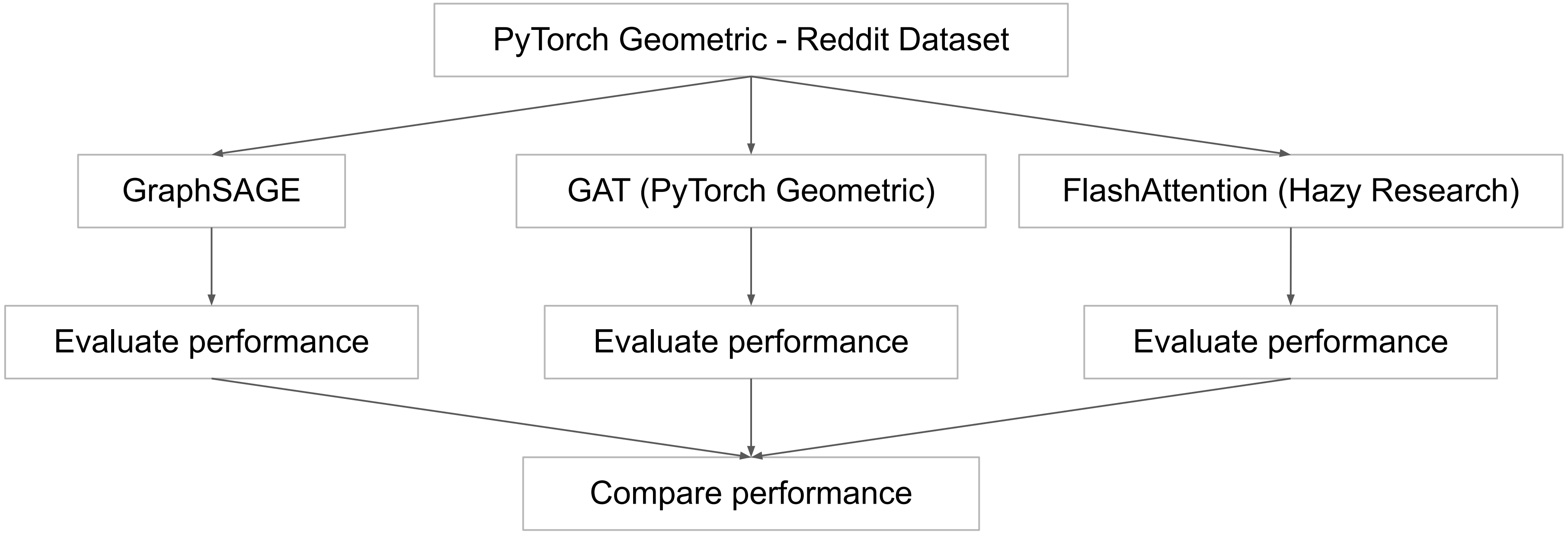
GraphSAGE | Graph Attention Networks (GAT) | FlashAttention

UCLA DataRes Research, Winter 2023

By Junwon Choi & Steven Lu Chen

Project Structure

Goal: Predict the community of each post/comment based on the graph structure and features of post/comment. Evaluate different models appropriate for this node classification task.



About the work environment

Hardware:

GPU:	NVIDIA GeForce GPU (24.576 GB)
CPU:	AMD Ryzen Threadripper PRO 5955WX 16-Cores (x86_64 | 32-bit, 64-bit)
RAM:	62 GB total, 8 GB swap

Software:

Platform:	Jupyter Notebook | VS Code (Remote - SSH Extension)
Language/Environment:	Python 3.10 / Conda
Main packages/tools:	torch, pyg, numpy

Remote workstation sponsored by: AI Safety at UCLA

GNNs: Spectral vs. non-Spectral Approach

Spectral	**Features**	**non-Spectral**
Uses spectral decomposition of graph Laplacian	**Convolutional Operation**	Uses a neighborhood aggregation operation
Uses eigenvalues & eigenvectors of graph Laplacian to encode graph structure	**Graph Structure Encoding**	Relies on node features & connectivity information
Offers *global* perspective of graph structure	**Global vs. Local Perspective**	More flexible in modeling *local* graph structure
Computationally efficient for regular graph structures	**Efficiency**	Can handle irregular graph structures
Generalized better to unseen graphs with similar structural properties	**Generalization**	Generalizes better to graphs with different structures

About the dataset: `torch_geometric.datasets.reddit`

Collection of Reddit posts and associated comments, represented as a directed graph.

Nodes: 232,965

- ❑ Represent posts/comments

Edges: 114,615,892 (weighted)

- ❑ Represent relationships between nodes (e.g. a comment responding to a post)

Features: 602

- ❑ Associated features of each node:
  - ❑ (e.g. text of post/comment and met)
  - ❑ Metadata (e.g. author, subreddit name)

Classes: 41

- ❑ Each corresponds to a specific community already categorized by PyTorch Geometric

Models Used

GraphSAGE

Graph Attention Networks (GAT)

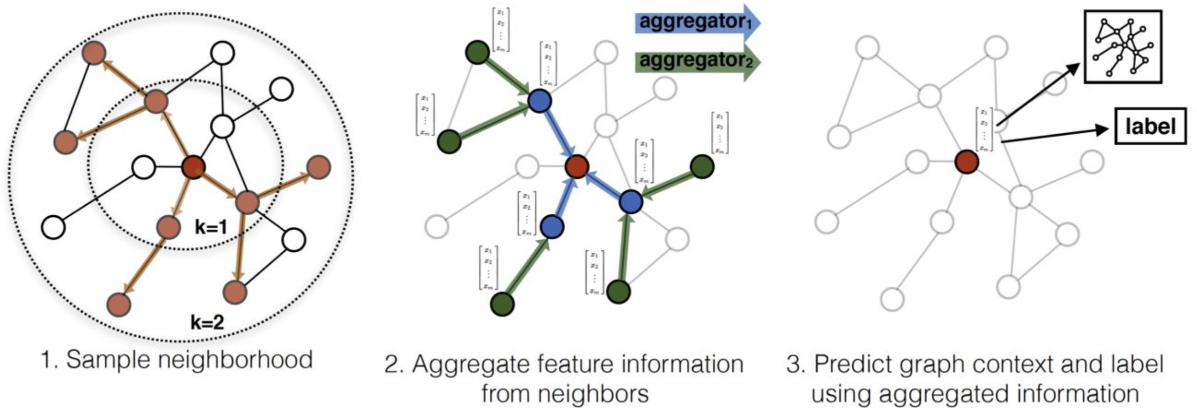
GraphSAGE

Type of framework for **inductive representation learning** on large graphs (>100,000 nodes)

GraphSAGE leverages node **feature information** to generate node embeddings for previously unseen data, instead of training individual embeddings for each node; i.e., embeddings are generated through sampling and aggregating features from the **local neighborhood** of each node.

Unlike previous node embedding approaches, GraphSAGE allows us to train GNNs even if individual nodes or portions of the graph are unseen.

GraphSAGE



k=1
k=2
1. Sample neighborhood
aggregator1
aggregator2
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

GraphSAGE

Paper Results:

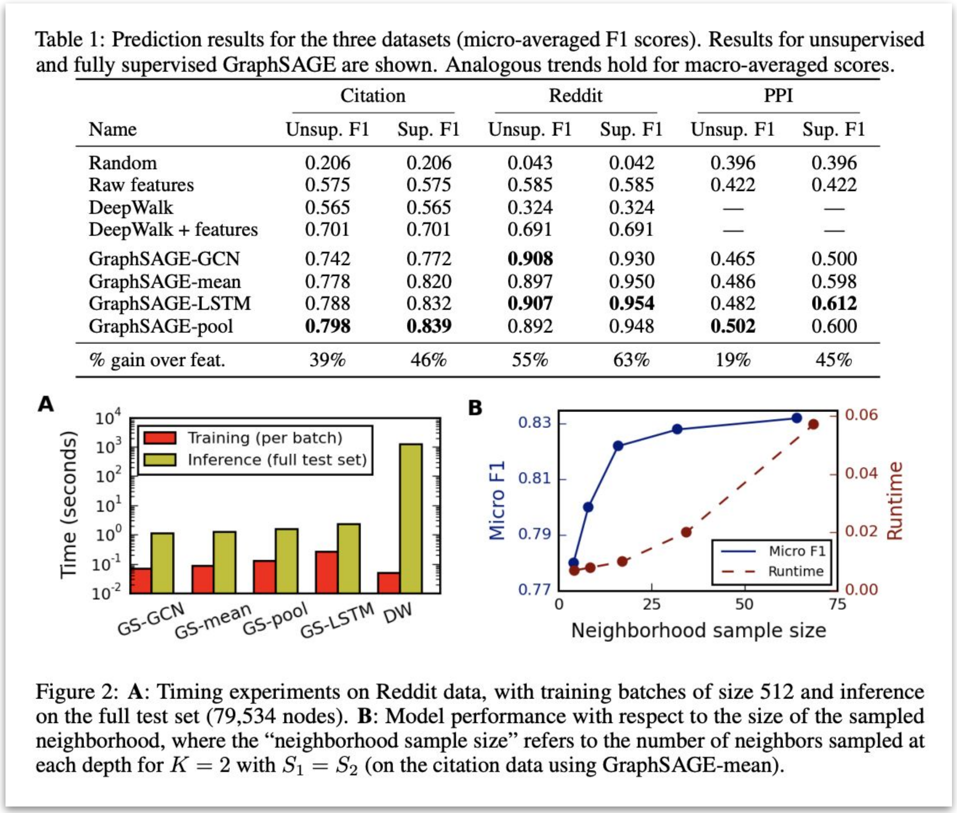


Table 1: Prediction results for the three datasets (micro-averaged F1 scores). Results for unsupervised and fully supervised GraphSAGE are shown. Analogous trends hold for macro-averaged scores.

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

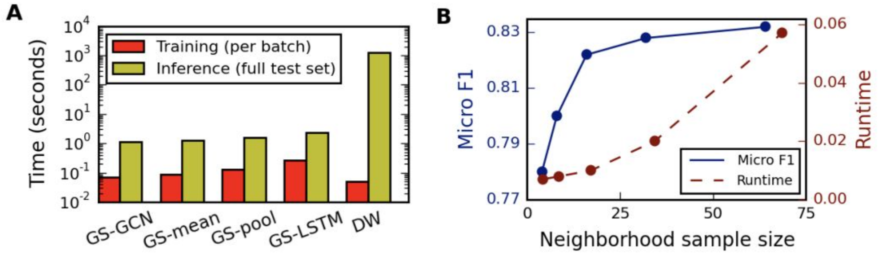


Figure 2: **A**: Timing experiments on Reddit data, with training batches of size 512 and inference on the full test set (79,534 nodes). **B**: Model performance with respect to the size of the sampled neighborhood, where the "neighborhood sample size" refers to the number of neighbors sampled at each depth for $K = 2$ with $S_1 = S_2$ (on the citation data using GraphSAGE-mean).

Graph Attention Networks (GAT)

Type of non-spectral GNN that uses **self-attention mechanisms** to weight the contribution of neighboring nodes during message passing.

Modification of Graph Convolutional Networks (GCNs).

Instead of using a fixed-weight filter to aggregate information from neighbors, GATs use self-attention mechanisms to to **learn a different weight for each neighbor node** based on feature representation of nodes.

GATs can **selectively attend** to most relevant neighbors for each node, which useful in graphs with complex structures.

Graph Attention Networks (GAT) - Architecture

Attention Coefficient

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

$$\downarrow$$

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

$$\downarrow$$

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

Normalized Attention Coefficient

Graph Attention Networks (GAT) - Architecture

$\vec{h}_i' =$ Updated representations of the *ith* node (output feature)

α_{ij} is used to compute linear combination of features corresponding to them

$$\vec{h}_i' = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

multi-head attention + concatenation

$$\vec{h}_i' = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

averaging used for final layer (prediction)

$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

Graph Attention Networks (GAT)

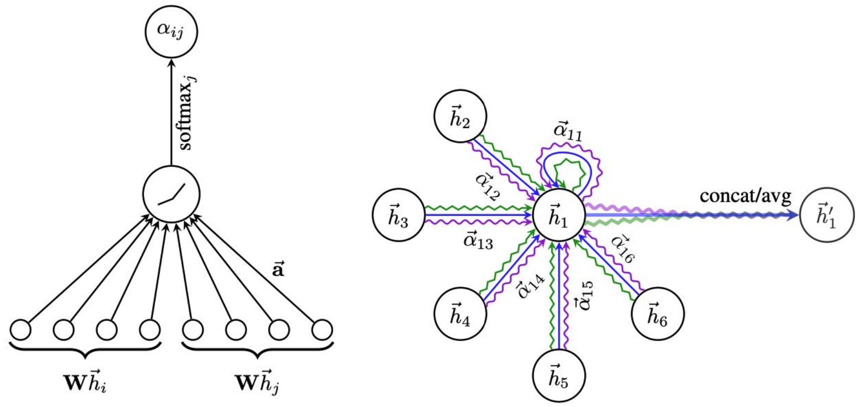


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Graph Attention Networks (GAT)

Paper Datasets:

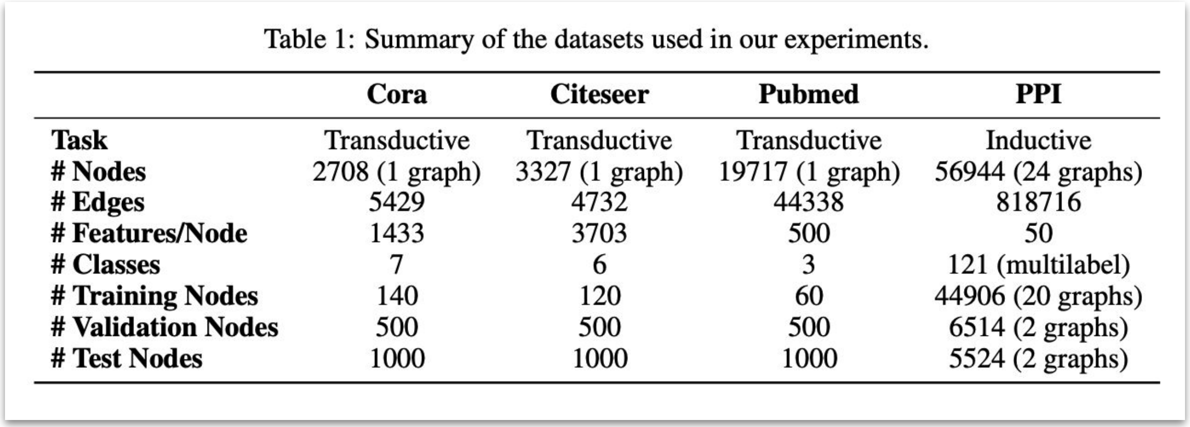


Table 1: Summary of the datasets used in our experiments.

	Cora	**Citeseer**	**Pubmed**	**PPI**
Task	Transductive	Transductive	Transductive	Inductive
# Nodes	2708 (1 graph)	3327 (1 graph)	19717 (1 graph)	56944 (24 graphs)
# Edges	5429	4732	44338	818716
# Features/Node	1433	3703	500	50
# Classes	7	6	3	121 (multilabel)
# Training Nodes	140	120	60	44906 (20 graphs)
# Validation Nodes	500	500	500	6514 (2 graphs)
# Test Nodes	1000	1000	1000	5524 (2 graphs)

Graph Attention Networks (GAT)

Paper Results:

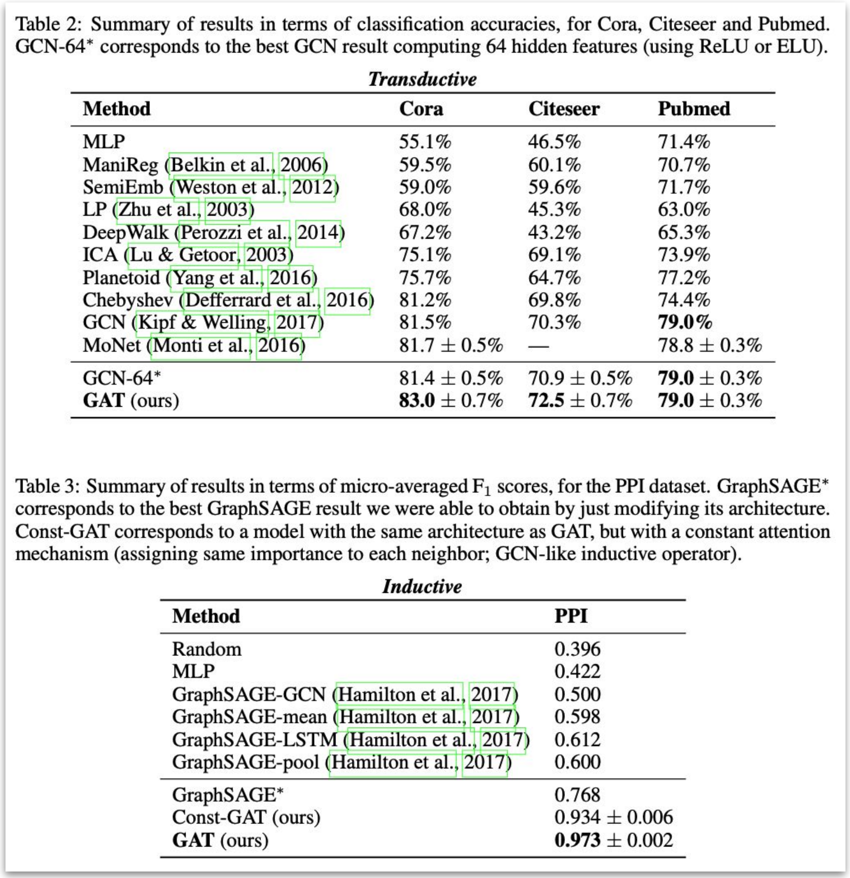


Table 2: Summary of results in terms of classification accuracies, for Cora, Citeseer and Pubmed. GCN-64* corresponds to the best GCN result computing 64 hidden features (using ReLU or ELU).

Transductive

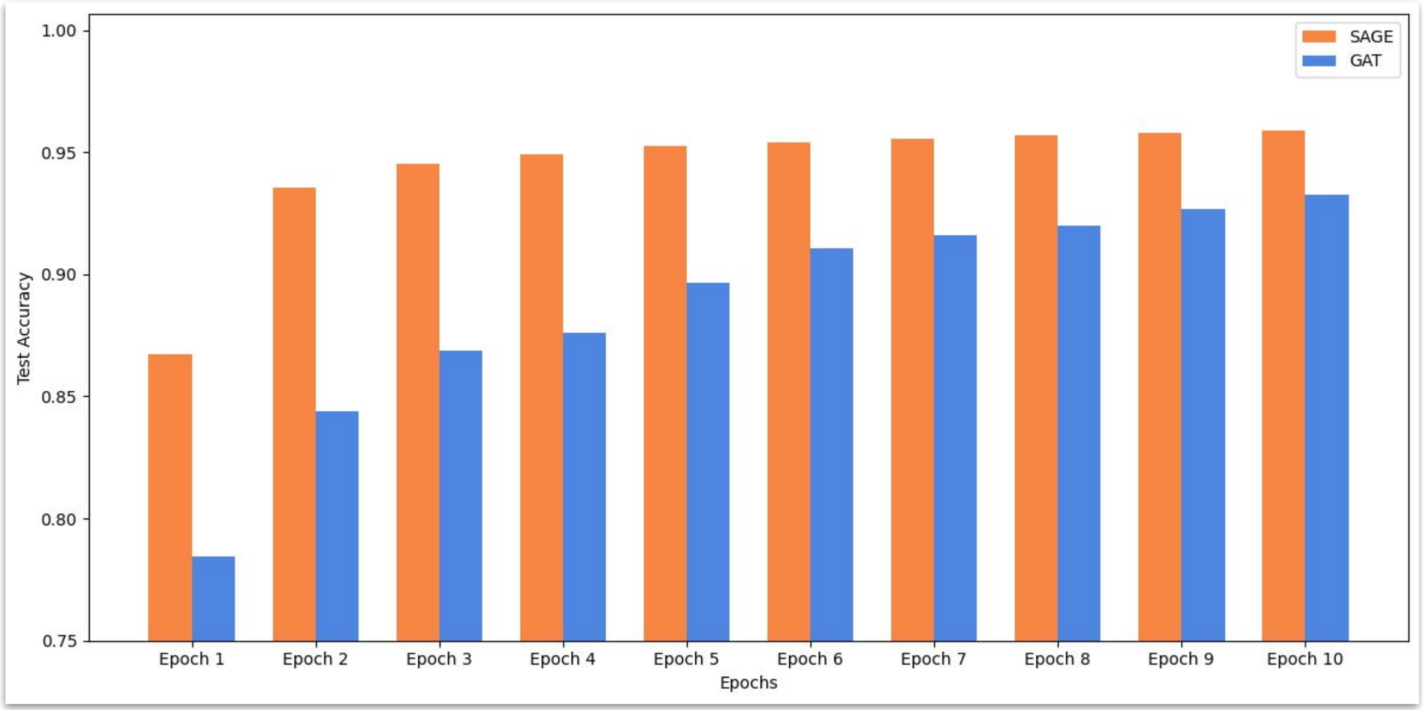
Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Table 3: Summary of results in terms of micro-averaged F_1 scores, for the PPI dataset. GraphSAGE* corresponds to the best GraphSAGE result we were able to obtain by just modifying its architecture. Const-GAT corresponds to a model with the same architecture as GAT, but with a constant attention mechanism (assigning same importance to each neighbor; GCN-like inductive operator).

Inductive

Method	**PPI**
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	**0.973** ± 0.002

GraphSAGE vs. Graph Attention Networks



GraphSAGE vs. Graph Attention Networks

Potential reasons for difference in performance:

- SAGE is more efficient in aggregating neighbor information
  - SAGE model computes node representations by aggregating the representations of its neighbors using a mean or max-pooling operation.
  - GAT model uses an attention mechanism to weight the neighbor representations, which may not be as efficient when the neighborhood sizes are large.
- SAGE can better capture local graph structure
  - SAGE model aggregates neighbor representations by performing a fixed number of message-passing steps.
  - GAT model uses an attention mechanism to weight the neighbor representation.

GraphSAGE vs. Graph Attention Networks

Potential reasons for difference in performance:

- ❏ PyTorch Geometric Reddit dataset may be better suited for SAGE
  - ❏ If the graph has many small subgraphs, the SAGE model may be better at capturing the local structure within each subgraph
- ❏ GAT’s hyperparameters may not be optimized for the PyTorch Geometric Reddit dataset
  - ❏ GAT model has several hyperparameters that can affect its performance, such as the number of attention heads, the hidden dimension size, and the dropout rate

Next steps

- ❏ Apply FlashAttention
- ❏ Community Detection
  - ❏ Challenge: Find and configure memory-efficient methods to identify and visualize groups in graph data
    - ❏ NetworkX
    - ❏ graph tool
- ❏ Network evolution

FlashAttention by Hazy Research at Stanford

Uses self-attention mechanisms similarly to GATs, but aims to further reduce memory usage by a process called **tiling**.

FlashAttention is **IO-aware**, which means it accounts for reads and writes across various levels of GPU memory. Overall, it requires fewer accesses to the GPU's high bandwidth memory (HBM) and is optimized for a variety of static RAM sizes.

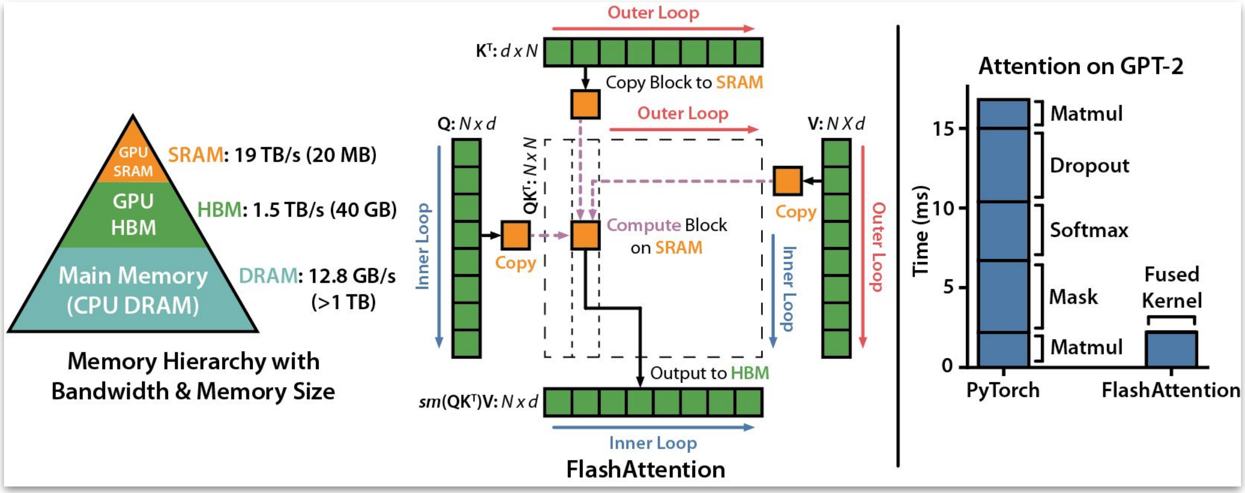
Compared to more primitive attention mechanisms, FlashAttention boasts:

❑ **faster model training**

❑ **higher quality models**

❑ **greater memory efficiency**

FlashAttention by Hazy Research at Stanford



FlashAttention by Hazy Research at Stanford

Faster model training:

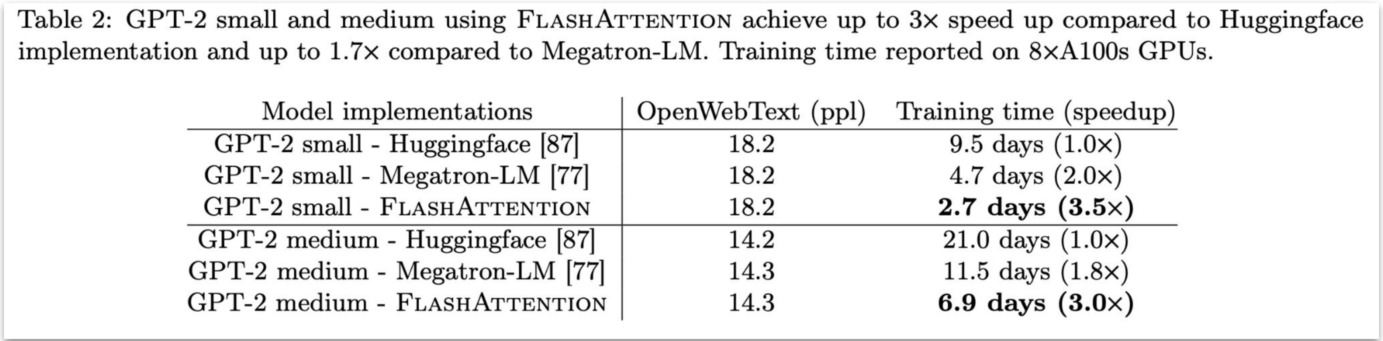


Table 2: GPT-2 small and medium using FlashAttention achieve up to 3× speed up compared to Huggingface implementation and up to 1.7× compared to Megatron-LM. Training time reported on 8×A100s GPUs.

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

FlashAttention by Hazy Research at Stanford

Higher quality models:

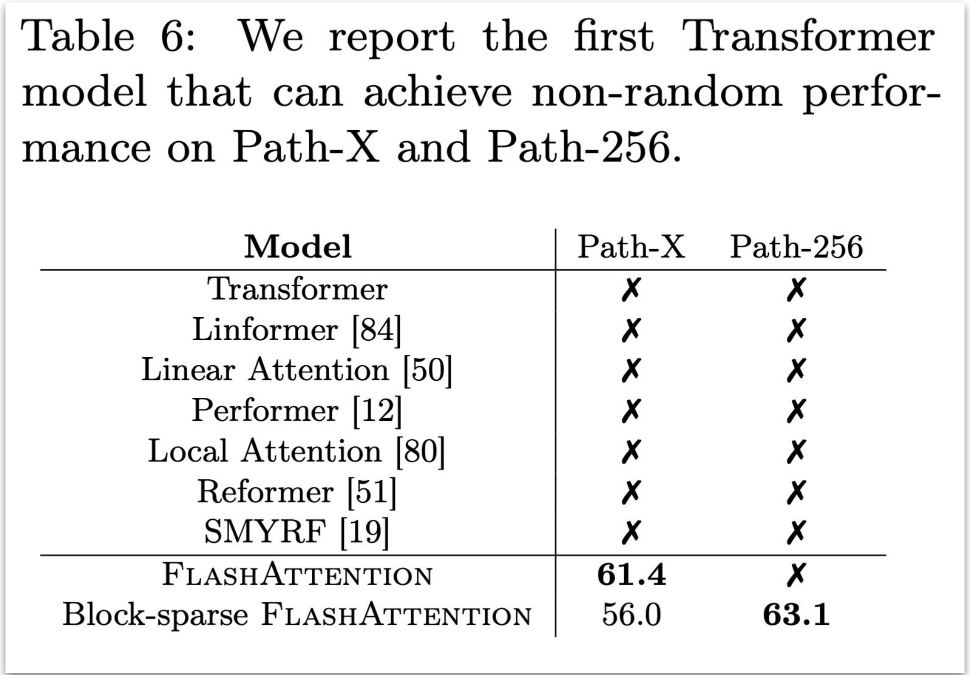


Table 6: We report the first Transformer model that can achieve non-random performance on Path-X and Path-256.

Model	Path-X	Path-256
Transformer	✗	✗
Linformer [84]	✗	✗
Linear Attention [50]	✗	✗
Performer [12]	✗	✗
Local Attention [80]	✗	✗
Reformer [51]	✗	✗
SMYRF [19]	✗	✗
FlashAttention	**61.4**	✗
Block-sparse FlashAttention	56.0	**63.1**

FlashAttention by Hazy Research at Stanford

Reduced memory usage:

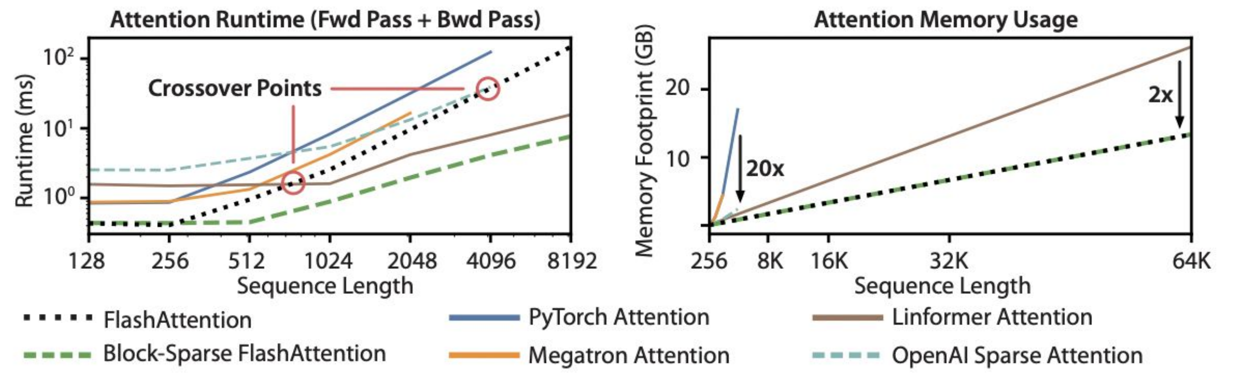


Figure 3: **Left:** runtime of forward pass + backward pass. **Right:** attention memory usage.

References

1. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. arXiv preprint arXiv:1710.10903 [stat.ML], 2018.

2. Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv preprint arXiv:2205.14135v2 [cs.LG], 2022

3. William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. arXiv preprint arXiv:1706.02216v4 [cs.SI], 2018

4. https://github.com/pyg-team/pytorch_geometric