From GNNs to Sparse Transformers: Graph-based architectures for Multi-hop Question Answering

Shane Acton[0000−0002−9035−4587] and Jan Buys[0000−0003−1994−5832]

Department of Computer Science, University of Cape Town, South Africa
ACTSHA001@myuct.ac.za,jbuys@cs.uct.ac.za

Abstract. Sparse Transformers have surpassed Graph Neural Networks (GNNs) as the state-of-the-art architecture for multi-hop question answering (MHQA). Noting that the Transformer is a particular message passing GNN, in this paper we perform an architectural analysis and evaluation to investigate why the Transformer outperforms other GNNs on MHQA. We simplify existing GNN-based MHQA models and leverage this system to compare GNN architectures in a lower compute setting than token-level models. Our results support the superiority of the Transformer architecture as a GNN in MHQA. We also investigate the role of graph sparsity, graph structure, and edge features in our GNNs. We find that task-specific graph structuring rules outperform the random connections used in Sparse Transformers. We also show that utilising edge type information alleviates performance losses introduced by sparsity.

Keywords: Transformers · Graph Neural Networks · Question Answering.

1 Introduction

Multihop question answering (MHQA) is a challenging language understanding task which involves reasoning over multiple facts often found across multiple documents [17,19]. Standard sequence models such as LSTM-based models have struggled on this task due to the need to model long-distance dependencies and to potentially perform multiple reasoning steps to answer a question.

Facts that are required to answer a question are often linked via common entities in what is called a reasoning chain [5]. Various MHQA works [7,14,9] demonstrate that MHQA input can be represented as an entity graph, and that Graph Neural Networks (GNNs) can be used to encode the graphs and learn to do reasoning to answer multihop questions. On tasks that involve long sequences, the Transformer [15] has been shown to exhibit higher task performance than previous approaches. However, as the Transformer processes information by performing full self-attention between all pairs of input tokens, its compute requirements scale quadratically with the number of tokens. This makes modeling very long token sequences with full self-attention Transformers intractable.

Sparse Transformers such as the Longformer [2] and Big Bird [20] perform self-attention between a subset of input token pairs. This allows them to process longer token sequences, at the cost of theoretical model expressiveness [20]. Sparse Transformers use rules to construct adjacency matrices that define which token pairs should communicate. This is conceptually identical to the function of a GNN, which makes use of an adjacency matrix to communicate node state information around a graph. GNNs, however, are a much more general class of models than Transformers. Sparse Transformer models are currently state-of-the-art on the WikiHop MHQA dataset [17], although a GNN-based model [9] is competitive with Sparse Transformers on the HotpotQA dataset [19].

In this paper, we aim to improve the performance of entity graph-based GNN models on the WikiHop dataset by leveraging the observation that the Transformer is a specific instance of a GNN in the message-passing framework. Typically, Transformer models operate at the token level only. However, we show that the GNN architecture can be substituted with a Transformer without changing any other part of the MHQA pipeline. The GNN-based MHQA pipeline involves encoding documents independently before aggregating token representations into course grained nodes. These nodes correspond to token spans in the document and can represent entities, sentences, and even whole documents. Finally, these node representations are encoded by a GNN [14,9], before being used to predict an answer. Despite the success of the Transformer, to the best of our knowledge, it has not previously been evaluated in such a GNN-based MHQA setting.

We consider a set of architectural design choices in GNN-based MHQA models, incorporating best practices established in the Transformer literature. We focus on three aspects in particular. First, we compare additive attention and scaled dot product (SDP) attention as used by Graph Attention Neural Networks (GAT) [16] and the Transformer respectively. We find that while both forms of attention perform favourably when compared to a non-attention baseline, SDP attention results in our best task performance.

Second, we evaluate the use of the Transformer's residual connections and position-wise Feed Forward Neural Networks (FFNN) in-between self-attention layers, which we refer to as the Transformer Update Function (TUF). We compare this to the position-wise gating mechanism common in GNNs [7,14]. We find that while gating has mixed effects on task performance, the TUF improves performance in all tested settings. Furthermore, we find that SDP attention pairs especially well with the TUF.

Third, we investigate graph structure and edge information in our GNN-based MHQA model. Full self-attention outperforms task-specific sparse attention when no edge information is available, supporting the idea that sparsity degrades model expressiveness [20]. However, when edge information is available, our task-specific sparse attention model outperforms the full self-attention model, indicating that edge information alleviates some disadvantages of sparse models.

2 Background

Graph Neural Networks (GNNs) were introduced to generalise Convolutional Neural Networks (CNNs) to non-Euclidean data [18]. GNNs have been applied widely due to the generality of the data they can model, and their natural modeling of sparsity. In this work we consider GNNs that can be defined in the message passing framework, which are able to operate on graphs of arbitrary topology [18]. After introducing this framework, we use it to define the Graph Attention Neural Network (GAT) [16], and the Transformer [15].

2.1 Message Passing GNNs

A graph can be represented by a set of n nodes and an edge matrix $E \in \mathbb{Z}^{n \times n}$ whose entries represent the edge types between any two nodes, with $E_{ij} = 0$ indicating that nodes i and j are unconnected. We use the notation given in the following equations to describe the general form of updating a single node state ($\boldsymbol{h}_i \in \mathbb{R}^f$) in the k^{th} message passing GNN (MP-GNN) layer.

1. Message:

$$M_{ij}^k = \phi^k(\boldsymbol{h}_i^k, \boldsymbol{h}_j^k, E_{ij}), \tag{1}$$

2. Aggregate:

$$A_i^k = \sum_{j \epsilon N(i)} \omega^k(M_{ij}^k, \boldsymbol{h}_i^k, \boldsymbol{h}_j^k), \tag{2}$$

3. Update:

$$\boldsymbol{h}_i^{k+1} = \gamma^k(\boldsymbol{h}_i^k, A_i^k). \tag{3}$$

Here ϕ^k is the message function at layer k and M_{ij}^k is its output which represents the message vector being passed to node i from node j along a single edge E_{ij}. ω^k is the aggregate function which prepares the message M_{ij}^k to be summed with all messages bound for node i. This sum forms a single aggregated message A_i^k to be passed to node i, which represents information from $N(i)$, the set of node i's neighbour nodes as defined by the edge matrix E, represented by their node indices. γ^k is the update function which combines the aggregated message A_i^k with the nodes current state $\boldsymbol{h}_i^k$ to create the node's state at the next layer, $\boldsymbol{h}_i^{k+1}$. The full algorithm for using these equations to encode a set of node states is given as Algorithm 1.

2.2 Attention

In order to define the GAT and Transformer as MP-GNNs we first define their attention mechanisms, restricted to the case of a single head for clarity. Additive attention involves the use of a multilayer perceptron (MLP) to predict *compatibility scores* between pairs of vectors:

$$\mathrm{MLP}(\boldsymbol{h}_i, \boldsymbol{h}_j) = \sigma(\mathrm{Concat}_f(\mathrm{W}_v \boldsymbol{h}_i; \mathrm{W}_v \boldsymbol{h}_j)\mathrm{W}_\mathrm{a}), \tag{4}$$

Algorithm 1 MP-GNN Node Encoding Procedure

Require:
Initial node features $\{\boldsymbol{h}_1^1, .., \boldsymbol{h}_n^1\}$
Edge matrix $E \in \mathbb{Z}^{n \times n}$

for all $k \in \{1, .., L\}$ **do**
  for all $i \in \{1, .., n\}$ **do**
    $A_i^k \leftarrow Zeros(f) \in \mathbb{R}^f$
    for all $j \in N(i)$ **do**
      $M_{ij}^k \leftarrow \phi^k(\boldsymbol{h}_i^k, \boldsymbol{h}_j^k, E_{ij})$
      $A_i^k \leftarrow A_i^k + \omega^k(M_{ij}^k, \boldsymbol{h}_i^k, \boldsymbol{h}_j^k)$
    end for
    $\boldsymbol{h}_i^{k+1} = \gamma^k(\boldsymbol{h}_i^k, A_i^k)$
  end for
end for
return $\{\boldsymbol{h}_0^L, .., \boldsymbol{h}_n^L\}$

where W_a and W_v are model weights and Concat_f denotes concatenation along the feature dimension. Finally, σ is a non-linear activation function.

Scaled dot product (SDP) attention calculates compatability scores as:

$$\mathrm{Dot}(\boldsymbol{h}_i, \boldsymbol{h}_j) = \frac{(\boldsymbol{h}_i \mathrm{W}_q)(\boldsymbol{h}_j \mathrm{W}_k)^T}{\sqrt{f}}, \tag{5}$$

where W_q and W_k are model weights for queries and keys respectively.

The compatibility scores S_{ij} are computed using either MLP(.) or Dot(.) for every pair of nodes i and j. The softmax function is used to normalise the compatibility scores and produce *attention scores*:

$$\alpha(\boldsymbol{h}_i, \boldsymbol{h}_j) = \mathrm{softmax}_j(S_{ij}) = \frac{exp(S_{ij})}{\sum_{k=1}^{l} exp(S_{ik})}. \tag{6}$$

To distinguish the use of additive or SDP attention in our models, we denote the GAT's additive attention as α_g and the Transformer's SDP attention as α_t.

2.3 GAT

We define the message passing equations for the GAT, in the case of a single attention head.

1. Message:

$$\phi^k(\boldsymbol{h}_i^k, \boldsymbol{h}_j^k, E_{ij}) = \mathrm{W_v} \boldsymbol{h}_j, \tag{7}$$

2. Aggregate:

$$\omega^k(M_{ij}^k, \boldsymbol{h}_i^k, \boldsymbol{h}_j^k) = \alpha_g(\boldsymbol{h}_i^k, \boldsymbol{h}_j^k) M_{ij}^k, \tag{8}$$

3. Update:

$$\gamma^k(\boldsymbol{h}_i^k, A_i^k) = \sigma(A_i^k). \tag{9}$$

Here, $\mathrm{W_v} \in \mathbb{R}^{f \times f}$ is a learned linear transformation. The additive attention function α_g is used in the aggregate step.

2.4 Transformer

While not typically thought of as a GNN, the Transformer [15] in fact performs message passing over the elements inputted to it. These elements are typically referred to as tokens, however they can naturally be considered as nodes. The standard Transformer performs full self-attention, meaning all nodes are connected to all nodes. Using the Transformer to model sparse graphs is an implementation detail, but typically would involve masking out the attention matrix with an adjacency matrix. This allows the Transformer to operate over arbitrary graph topology. The Transformer is described using the message passing notation using the equations that follow.

1. Message:

$$\phi^k(\boldsymbol{h}_i^k, \boldsymbol{h}_j^k, E_{ij}) = \mathrm{W_v}\boldsymbol{h}_j, \tag{10}$$

2. Aggregate:

$$\omega^k(M_{ij}^k, \boldsymbol{h}_i^k, \boldsymbol{h}_j^k) = \alpha_t(\boldsymbol{h}_i^k, \boldsymbol{h}_j^k)M_{ij}^k, \tag{11}$$

3. Update:

$$\gamma^k(\boldsymbol{h}_i^k, A_i^k) = \mathrm{TUF}(\boldsymbol{h}_i^k, A_i^k). \tag{12}$$

Here the SDP attention function α_t is used. The other difference is that the Transformer uses the TUF update function introduced in the original Transformer paper [15]. The Transformer Update Function (TUF) consists of residual connections, Layernorms [1], and a Feed Forward Neural Network (FFNN), formally given by equations:

$$\mathrm{TUF}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \mathrm{Norm}(\boldsymbol{x}' + \mathrm{FFNN}(\boldsymbol{x}')), \tag{13}$$

$$\boldsymbol{x}' = \mathrm{Norm}(\boldsymbol{x}_1 + \boldsymbol{x}_2). \tag{14}$$

Norm is the LayerNorm [1] function, and FFNN is an MLP with a single hidden layer.

Both the Transformer and GAT make use of multi-headed attention. While the details differ slightly between the two models, the concept is the same. Multi-headed attention involves computing multiple attention scores for every pair of nodes using distinctly parameterised functions (heads). These multiple attention scores are then used to perform multiple weighted sums which are then combined. Making use of multiple attention heads increases the theoretical expressiveness of the attention mechanism [15], possibly by allowing different heads to focus on different criteria when comparing vectors [6]. Multi-headed attention is compatible with the message passing framework, but for brevity we omit its definitions. When the Transformer is viewed as an MP-GNN, the multi-headed attention block is the GNN's message and aggregate functions, while the rest of the operations all fall under the GNN's update function.

2.5 Gating and over-smoothing

MHQA GNN's commonly [14,7] employ some form of a gating [13] mechanism. Gating can be used in a GNN's update function to modulate how neighbour node information updates node states. We consider a particular gating function common in GNN-based MHQA models [7,14], defined as:

$$\mathrm{G}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \tanh(\boldsymbol{u}) \odot \boldsymbol{g} + \boldsymbol{x}_1 \odot (1 - \boldsymbol{g}), \tag{15}$$

$$\boldsymbol{u} = \boldsymbol{x}_1 \mathrm{W}_\mathrm{u} + \boldsymbol{x}_2, \tag{16}$$

$$\boldsymbol{g} = \sigma(\mathrm{Concat}_f(\boldsymbol{x}_1, \boldsymbol{u})\mathrm{W}_\mathrm{g}), \tag{17}$$

where $\odot$ denotes element-wise multiplication. $\mathrm{W}_\mathrm{g} \in \mathbb{R}^{2f \times f}$ and $\mathrm{W}_\mathrm{u} \in \mathbb{R}^{f \times f}$ are learned matrices. Using gating this way in a GNN's update function is believed to reduce the over-smoothing problem [14]. Over-smoothing prevents the effective use of deep GNNs. It is thought to be caused by the loss of node identities as node representations become more similar to each other every layer [4].

Although the Transformer is an MP-GNN, it does not suffer from the over-smoothing problem, with extremely deep Transformers like GPT3 [3] (96 layers) producing state-of-the-art results. This may be due to the use of residual connections in the TUF, which are not commonly used in GNN-based MHQA models. Thus, in this work we investigate whether the TUF and gating are complimentary, or if one is superior to the other.

3 Model

We introduce our Graph Neural Network (GNN)-based Multihop Question Answering (MHQA) model. The model is trained to answer multiple-choice questions in the form of a question q, a set of text passages S_q containing information needed to answer the question, a set of answer candidates C_q, and the answer $a_q \in C_q$. For WikiHop, the question comes in the form of the semantic triple (Subject, Relationship, ?) where Subject is an entity and "?" the answer to be predicted. We use heuristic rules proposed by previous work [14] to construct graphs based on the question inputs. The model uses a pre-trained token embedder such as GloVe [11] or BERT [8], alongside a parameterised Transformer-encoder to create graph node embeddings. The graphs are then encoded by the GNN, and the graph encodings are used to predict the answer to the question.

3.1 Graph Construction

We follow the Heterogeneous Document Entity model (HDE) [14] to construct a graph composed of entities, answer candidates, and documents for a given multiple-choice question. These graphs are heterogeneous in that they contain multiple node and edge types. They also include information at multiple levels of granularity, namely the phrase-level entity and candidate nodes, and the course-grained document nodes. An example is shown in Figure 1.

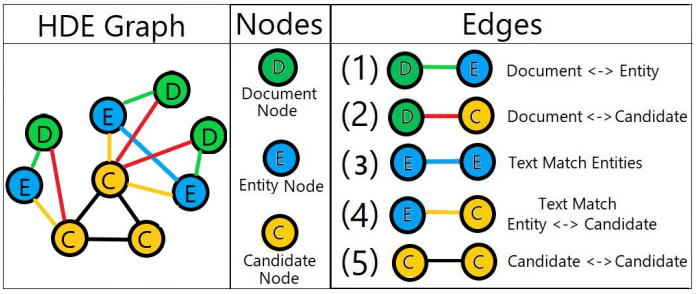

Fig. 1. An example graph used in our model, similar to HDE graphs [14], shown here together with node and edge types.

Entity nodes represent token spans in containing documents. The entities considered by our model are only those which are either a string match with any of the answer candidates, or the subject entity found in the question. There is an entity node for each extracted entity mention in each document. Therefore there may be multiple entity nodes representing identical text from different locations: these are referred to as co-mentions. The other node types are: a document node for each supporting document, and a candidate node for each answer candidate.

A set of heuristic rules connect nodes with the following edge types:

1. Document nodes connect to entity nodes extracted from the document.
2. Document nodes are also connected to candidate nodes if their text is found in the document.
3. *Comention Edge*: Entity comentions are connected. This edge type enables connecting mentions of the same entity across long distances in the input.
4. Entity nodes are connected to the candidate node whose text it matches.
5. All candidate nodes are connected to each other.

3.2 Graph Node Embedding

The first step in constructing embeddings for each of the graph nodes is to generate token embedding sequences for each document S_q^i, candidate C_q^j, and the query q. The token embeddings are obtained by use of a pre-trained token embedder such as GloVe or BERT. This involves first breaking the text up into tokens recognised by the embedder's vocabulary, and then using the embedder to produce a vector for each token. This yields *token matrices* X_q, X_s^i, and X_c^j for the query, i^{th} document, and j^{th} candidate respectively. Beyond this point, the model architecture is not affected by the choice of token embedder.

Each node represents a token span, i.e., it corresponds to a subsequence of one of the token matrices. We refer to these subsequences of token matrices as *node matrices*, and denote the node matrix for a node with index t as $X_t \in \mathbb{R}^{l \times f}$ for a node with l tokens within its span.

We concatenate the query token matrix X_q with each node matrix X_t, and encode the result with a Transformer encoder Trans_c:

$$X_{qt} = \text{Trans}_\text{c}(\text{Concat}_s(X_q, X_t)), \tag{18}$$

where Concat_s denotes concatenation along the sequence dimension. The output of the Transformer is a query-aware matrix X_{qt}. We use the vector corresponding to the first token in the sequence X_{qt} as our initial node embedding $\mathbf{h}_t^0$.

3.3 GNN Encoding

We use message passing GNNs with L-layers to encode the graphs. We consider several GNN variations: We compare additive attention, SDP attention and a GNN without attention. We also consider using the Transformer Update Function (TUF), gating, both, or neither.

GNN Cores with Edge Types We introduce the term *GNN Core* to refer to the choice of message and aggregate functions in a GNN. All our GNNs include edge type embeddings. Given the edge matrix $E \in \mathbb{Z}^{n \times n}$ such that E_{ij} represents the edge type between nodes i and j, the edge embeddings are defined as

$$V_{ij} = \text{EdgeTypeEmb}(E_{ij}) \in \mathbb{R}^f, \tag{19}$$

where EdgeTypeEmb maps each unique edge type to a learned f-dimensional vector.

We define three GNN cores:

1. SDP-Att Core. The SDP attention mechanism with edge type embeddings as described by the Relative Positional Embedding Transformer [12].
2. MLP-Att Core. A version of the GAT modified to use edge type embeddings with additive attention.
3. Mean Core. A version of a GNN without attention similar to previous GNN-based MHQA models, modified to use edge type embeddings.

All of our GNN Cores share the same message function, which applies a linear transformation to the node embedding and adds the edge type embedding:

$$\phi(\boldsymbol{h}_i^k, \boldsymbol{h}_j^k, E) = \text{W}_\text{v}\boldsymbol{h}_j^k + V_{ij}. \tag{20}$$

Next, we define the aggregate function for each of the GNN Cores.

i. Mean Core Aggregate:

$$\omega^k(M_{ij}^k, \boldsymbol{h}_i, \boldsymbol{h}_j) = \frac{M_{ij}^k}{|N(i)|}. \tag{21}$$

ii. SDP-Att Core Aggregate:

$$\omega^k(M_{ij}^k, \boldsymbol{h}_i, \boldsymbol{h}_j) = \alpha_t(\boldsymbol{h}_i, \boldsymbol{h}_j)M_{ij}^k. \tag{22}$$

iii. MLP-Att Core Aggregate:

$$\omega^k(M_{ij}^k, \boldsymbol{h}_i, \boldsymbol{h}_j) = \alpha_g(\boldsymbol{h}_i, \boldsymbol{h}_j)M_{ij}^k, \tag{23}$$

Table 1. Model hyperparameters for all experiments.

Hyper Parameter	Value
Training Epochs	30
Learning Rate (LR)	0.01
LR Schedule Exponential Decay	0.9
Dropout	0.1
Batch Size	1
Num GNN layers	9
Model Dimension	**300** for GloVe **512** for BERT
Num GNN Heads	**4** if attention-based

Update Functions As a GNN update function we consider the gating function G described in section 2.5, the TUF described in section 2.4, or composing them together as follows:

$$\gamma(\boldsymbol{h}_i^k, A_i^k) = \mathrm{G}(\boldsymbol{h}_i^k, \mathrm{TUF}(\boldsymbol{h}_i^k, A_i^k)). \tag{24}$$

It is also possible to use neither, as the canonical GAT [16] does.

3.4 Output Model

Finally, our model outputs a probability distribution over answer candidates C_q. The output of the L^{th} GNN layer is a set of encoded node states denoted $\mathbf{h}_i^L$ for the i^{th} node. For each candidate $c \in C_q$ the model extracts the set of all entity node vectors which correspond to candidate c: $E_c = \{\mathbf{h}_i^L \; \forall i \mid i \text{ is a mention of } c\}$.

Candidate score c_s is based on both entity and candidate node states:

$$c_s = \mathrm{MLP}_c(\mathbf{h}_c^L) + \max_{\mathbf{e} \in E_c}(\mathrm{MLP}_\epsilon(\mathbf{e})), \tag{25}$$

where MLP_c and MLP_ϵ are MLP's to score candidates and entities respectively. The final probability distribution over all candidates is obtained by performing a softmax over the candidate scores c_s.

4 Experimental Setup

We train and evaluate our models on the WikiHop MHQA dataset [17]. WikiHop follows the multiple choice MHQA structure described in section 3. Table 1 shows the hyperparameters used in all of our experiments. No model-specific hyperparameter tuning was performed for any of the results. The model (hidden state) dimension is determined by the dimensionality of the token embedder, i.e., 300 for GloVe-based models and 512 for BERT-based models. We use an LR scheduler with exponential decay [10]. In preliminary experiments mini-batching (using a batch size greater than 1) did not improve performance.

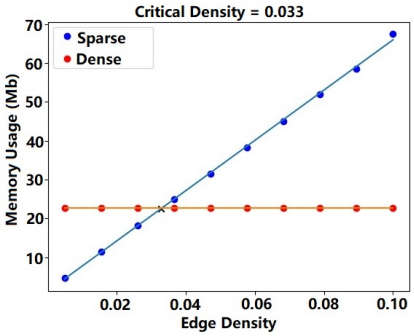

Fig. 2. Edge density vs. memory used for dense and sparse attention implementations. Tested with 500 graph nodes, 100 features per node, and 10 attention heads.

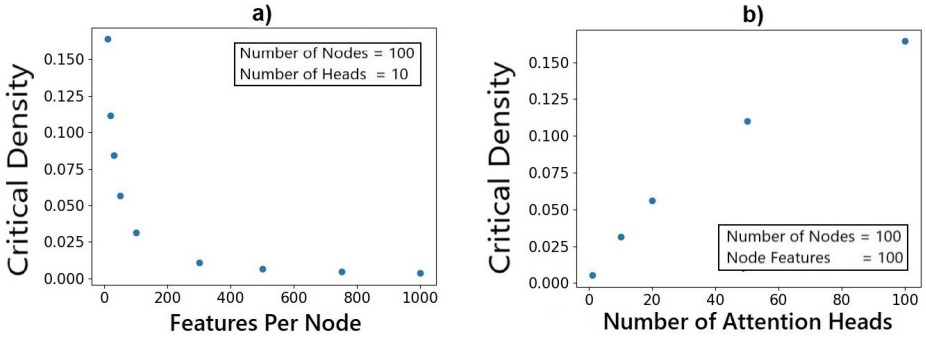

Fig. 3. Plots showing how the critical edge density changes while varying (a) the number of features per node, and (b) the number of attention heads.

4.1 Implementation

Our models are implemented in Pytorch. We use Pytorch Geometric[1] to implement our GAT and MLP-based GNNs. For our SDP GNNs we use the native Pytorch implementation of the Transformer's multi-headed attention. Tokenisation is either word-based to be compatible with the GloVe vocabulary, or performed using Huggingface's subword tokenisation utilities for our BERT-based models.[2]

Sparse and Dense GNN Implementations Pytorch Geometric is designed to be optimal for sparse graphs (the *sparse approach*). The Transformer implementation (the *dense approach*) is optimised for fully connected graphs, although it can also model sparse graphs. The sparse approach's memory requirements scale linearly with the density of the inputted graph, while the dense approach's

[1] https://pytorch-geometric.readthedocs.io/en/latest/
[2] https://huggingface.co/

Table 2. MHQA accuracies on the WikiHop development set for our best GNN configurations using GloVe or BERT embeddings, compared to the HDE (the best previous GNN-based model) and BigBird (the best-performing Transformer).

Model	Dev Accuracy
GNN (GloVe)	66.0
GNN (BERT)	71.4
HDE [14]	68.1
BigBird [20]	**75.9**

Table 3. WikiHop development set accuracies of our GloVe-based GNN models with different architectural choices.

	Gating	✗	✓	✗	✓
	TUF	✗	✗	✓	✓
GNN Core		Dev Accuracy			
Mean		52.7	58.5	62.5	60.2
MLP-Att		29.0	62.9	60.4	64.0
SDP-Att		13.0	61.4	**66.0**	64.7

memory requirements are independent of graph density. However, the dense approach requires less memory when operating on fully connected graphs.

We analyse the memory requirements of these two implementation approaches for SDP attention, comparing memory usage across graphs of varying edge densities. This allows us to find the critical edge density (hereafter referred to as the *critical density*) where the optimal implementation changes. Figure 2 shows that the critical density is 0.033, meaning only graphs with edge densities less than 0.033 should be implemented with the sparse approach.

We further investigate how varying the number of nodes, number of attention heads, and number of features per node affects the memory scaling of the two implementation approaches. We found that the number of nodes has no significant relationship with Critical Density. Figure 3 (a and b) shows that when the number of features is very small, or the number of heads is very large, the sparse approach may be preferable, but only for very sparse graphs. Therefore, in all but the most extreme cases the memory requirements of the dense approach will be lower. The PyTorch Geometric implementation is therefore suboptimal for tasks such as multi-hop question answering.

5 Results

All results presented in this section are accuracies on the WikiHop development set.[3] Table 2 shows the results of our best performing models when using GloVe and BERT as token embedders. These two models make use of the SDP-Att Core and the TUF without gating. The performance of our model with GloVe embeddings is lower than that of HDE [14]. HDE uses character n-gram embedding embeddings in addition to GloVe and a custom attention mechanism for getting node embeddings from token embeddings, which we substituted for a Transformer encoder. Using BERT as token embedder however, our model is more accurate than HDE, which is to date the highest performing GNN-based

[3] Evaluation on the hidden test set was not possible due to incompatible software versions on the evaluation portal.

Table 4. WikiHop development set accuracies of our GloVe-based GNNs, comparing sparse vs fully connected graphs, with or without edge embeddings.

Sparsity	Edge Embeddings	Dev Accuracy
✓	✓	**64.7**
✗	✓	64.4
✓	✗	60.2
✗	✗	61.5

Table 5. Task-specific vs random graph structure, with or without edge embeddings on the WikiHop development set, sing BERT-based GNNs

Structure	Edge Embeddings	Dev Accuracy
Task-Specific	✓	**68.2**
Task-Specific	✗	64.4
Random	✓	63.8
Random	✗	64.5

MHQA model on the WikiHop dataset. Our best performing model falls short of the performance of the BigBird Sparse Transformer [20]. BigBird encodes token sequences directly, and the full model is pre-trained with Masked Language Modeling (MLM), while in our approach only the BERT token encoder is pre-trained. BigBird makes use of random connections between tokens, in combination with some rule-based connections. However, it does not include edge type information to distinguish these distinct edges.

5.1 GNN Architecture

Table 3 shows the performance of our GNN-based MHQA model with various GNN Cores as well as different update functions (Gating and/or TUF). The results demonstrate a number of key findings. First, the attention-based GNN Cores (MLP-Att and SDP-Att) outperform the GNN without attention (Mean Core) in most settings. The attention-based GNNs also achieve better maximum task performances. Second, when neither gating nor the TUF is included, all evaluated GNN variants perform worse. TUF boosts task performance in all settings where it is included. Finally, the attention-based GNNs are especially reliant on the TUF and/or gating compared to the non attention-based GNN Core.

The Mean Core and SDP-Att Core models perform best when the TUF is used without gating. On the other hand, the combination of gating and the TUF is beneficial for MLP-Att Core. Thus we can draw no clear conclusion about the use of gating in our GNNs as these results may be due to noise.

5.2 Graph Structure and Edge Embeddings

Table 4 compares models which use sparse graphs against variants which use fully connected graphs (similar to the vanilla Transformer). The GNNs used in this experiment all use SDP-Att Core with gating and the TUF. Here sparse graphs are the rules-based task-specific graphs described in section 3. We construct fully connected graphs by starting with our sparse graphs, and adding in a new edge type *unconnected* which connects all unconnected nodes. Thus,

there is no loss in edge information when comparing our sparse and fully connected graphs, although we also evaluate without using edge embeddings. The results demonstrate that without edge information, using fully connected graphs performs better than sparse graphs. However, when including edge information, sparse graphs can produce similar performance.

Table 5 compares the use of task-specific graph structure based on rules developed in the GNN-based MHQA literature [14] against random sparse structure. In this experiment we use GNNs with SDP-Att Core including gating and the TUF, using BERT as token encoder. Here, the random structure involves replacing edges with random edges, defined by randomly selecting two nodes to connect. We do not replace (Candidate-Candidate) edges with random edges. This ensures that candidates remain fully connected to each other. This random shuffling preserves the number of edges.

In random graph structure with edge embeddings, the edge type is simply the tuple of the node types which it connects (e.g., Candidate-Entity). The results clearly show that task-specific structure in combination with edge information is important to model performance. Neither task-specific structure nor edge information alone significantly boosts model performance. This serves to validate the specific graph structuring rules which have been developed in the GNN-based MHQA literature [14].

6 Discussion

There are several limitations to how much can be claimed about the generalizability of our experimental results. Due to high computational requirements we only trained a single model per configuration, without model-specific hyperparameter tuning. We also only evaluated on a single dataset for which the test set was unavailable. However, we believe our results are nonetheless valuable and may spur further research into the connection between GNNs and Sparse Transformers.

Bigbird, the Sparse Transformer, [20] makes use of random connectivity to decrease the average number of connections between each node. Our results indicate that random connectivity should be replaced by problem-specific structure where possible. Our results also motivate the use of edge type information alongside the problem specific structure. Finally, our results demonstrate the value of using SDP attention in combination with the TUF, exactly as is done in the Transformer model [15]. Given the popularity of the GAT GNN, there may be many models which could benefit from (1) switching from additive attention to SDP attention, and (2) including the TUF in the GNN's update function. Essentially, this is a recommendation to replace existing attention-based GNNs with the Transformer. Our empirical analysis on memory requirements in section 4.1 also indicate that for graphs used in realistic Natural Language Processing scenarios, using a vanilla Transformer implementation is preferable to a message passing implementation which is optimised for sparse graphs.

Many token-level NLP models make use of end-to-end pre-training to prepare the model for the final task [8,20]. GNN-based NLP models, including ours, do not make use of end-to-end pre-training, instead relying only on pre-trained token embedders [14,9]. This motivates the future development of pre-training strategies for GNN-based NLP models. A graph-based model which performs on par with Sparse Transformers would reduce the memory requirements compared to token-level models.

7 Conclusion

We implemented and evaluated a simplified version of the GNN-based Heterogeneous Document Entity (HDE) MHQA model. We used the WikiHop dataset to evaluate two primary differences between the GAT and the Transformer, namely (1) the type of attention mechanism, and (2) the use of the Transformer Update Function (TUF) as an update function. Our results serve as a case study motivating the use of scaled dot product (SDP) attention and the TUF — essentially the canonical Transformer. We also investigate the role of graph sparsity, graph structure, and edge information in our MHQA model. Our results demonstrate the value in task-specific graph structure rules over random connectivity and fully connected graphs with an emphasis on the use of problem specific edge information. The results further indicate that without edge information, task-specific connection rules may not yield performance gains over random sparse connections or fully connected graphs. These insights may provide a path to improving token-level Sparse Transformer performance. Finally, our results show that there is room for further research to close the performance gap between token-level models and graph-level models with coarse-grained nodes.

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. stat **1050**, 21 (2016)
2. Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer. CoRR **abs/2004.05150** (2020), https://arxiv.org/abs/2004.05150
3. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
4. Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., Sun, X.: Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 3438–3445 (2020)
5. Chen, J., Lin, S., Durrett, G.: Multi-hop question answering via reasoning chains. CoRR **abs/1910.02610** (2019), http://arxiv.org/abs/1910.02610
6. d'Ascoli, S., Touvron, H., Leavitt, M.L., Morcos, A.S., Biroli, G., Sagun, L.: Convit: Improving vision transformers with soft convolutional inductive biases. In: Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. vol. 139, pp. 2286–2296. PMLR (2021), http://proceedings.mlr.press/v139/d-ascoli21a.html

7. De Cao, N., Aziz, W., Titov, I.: Question answering by reasoning across documents with graph convolutional networks. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). pp. 2306–2317 (2019)
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (1) (2019)
9. Fang, Y., Sun, S., Gan, Z., Pillai, R., Wang, S., Liu, J.: Hierarchical graph network for multi-hop question answering. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 8823–8838 (2020)
10. George, A.P., Powell, W.B.: Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. Machine learning **65**(1), 167–198 (2006)
11. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. pp. 1532–1543. ACL (2014). https://doi.org/10.3115/v1/d14-1162, https://doi.org/10.3115/v1/d14-1162
12. Shaw, P., Uszkoreit, J., Vaswani, A.: Self-attention with relative position representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers). pp. 464–468 (2018)
13. Sigaud, O., Masson, C., Filliat, D., Stulp, F.: Gated networks: an inventory. CoRR **abs/1512.03201** (2015), http://arxiv.org/abs/1512.03201
14. Tu, M., Wang, G., Huang, J., Tang, Y., He, X., Zhou, B.: Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. pp. 2704–2713 (2019)
15. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
16. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018)
17. Welbl, J., Stenetorp, P., Riedel, S.: Constructing datasets for multi-hop reading comprehension across documents. Transactions of the Association for Computational Linguistics **6**, 287–302 (2018)
18. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems (2020)
19. Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W.W., Salakhutdinov, R., Manning, C.D.: HotpotQA: A dataset for diverse, explainable multi-hop question answering. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018. pp. 2369–2380. Association for Computational Linguistics (2018). https://doi.org/10.18653/v1/d18-1259, https://doi.org/10.18653/v1/d18-1259
20. Zaheer, M., Guruganesh, G., Dubey, K.A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., Ahmed, A.: Big bird: Transformers for longer sequences. In: Advances in Neural Information Processing Systems 33 (2020), https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html