GraphVQA with Graph Transformer using Dependency Graph and AMR Graph

Itisha Yadav
st176482@stud.uni-stuttgart.de
Nianheng Wu
st172149@stud.uni-stuttgart.de
Institut für Maschinelle Sprachverarbeitung/ Universität Stuttgart

Abstract

This is a report for seminar Advanced Machine Learning at University of Stuttgart. We aim at GraphVQA task and adopted the original GraphVQA framework. Based on which we substituted the Seq2Seq module with a GNN based module to parse questions in this task. We further experimented our updated system with multiple variations, including using different graph construction schemes (dependency graphs and AMR graphs), using pretrained model BERT for better node representations, or using positional features and bidirectional edges in GNN. Our final results outperformed the original ones.

1 Introduction

Images provide a representation of relationship between its objects. Scene graphs are used to encode objects in the images and their pairwise relationships. In other words, scene graph is a structured representation of a scene that can clearly express the objects, attributes, and relationships between objects in the scene. As computer vision technology continues to develop, people are no longer satisfied with simply detecting and recognizing objects in images; instead, people look forward to a higher level of understanding and reasoning about visual scenes. For example, given an image, we want to not only detect and recognize objects in the image, but also understand the relationship between objects (visual relationship detection), and generate a text description (image captioning) based on the image content. Alternatively, we might want the machine to tell us what the little girl in the image is doing (Visual Question Answering (VQA)), or even remove the dog from the image and find similar images (image editing and retrieval), etc. These tasks require a higher level of understanding and reasoning for image vision tasks. Scene graph is just such a powerful tool for scene understanding. Therefore, scene graphs have attracted the attention of a large number of researchers, and related research is often cross-modal, complex, and rapidly developing. The paper (Chang et al., 2021) provides a comprehensive survey of Scene Graphs, their generation and applications. Although a lot of tasks around scene graphs has been proposed, Scene Graph QA is so far relatively under-explored.

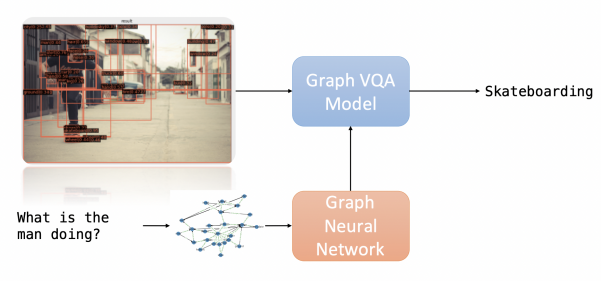

Figure 1: Task overview

In this report, we focus on Scene Graph Question Answering (GraphVQA), which uses graph representations called Scene Graphs instead of the raw images. GraphVQA is a graph neural network framework that translates and executes a natural language question as multiple iterations of message passing among graph nodes and supports question-answering on Scene Graphs. In the previous works, most of them experimented with fully-connected graphs (Hu et al., 2019), (Li et al., 2019), which unfortunately leaves out import structural information. A more recent work (Liang et al., 2021) used a de facto GNN system that achieved state-of-the-art performance with a big margin, but in their framework they adopted a seq2seq structure to process questions. The work in this paper is mainly based on this framework but further explore the power of a full GNN system by substitute the seq2seq module with a GNN based module.

The aim of the project is to explore different graph construction methods for natural language

[0]The code is in this repository: link

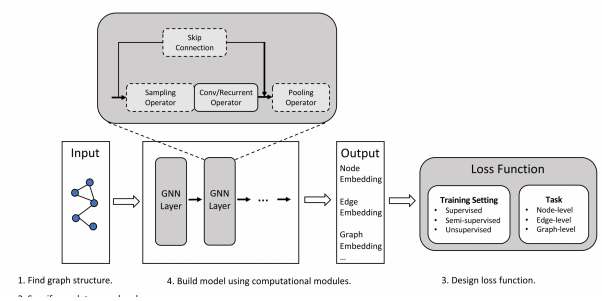

Figure 2: The general design pipeline for a GNN model.

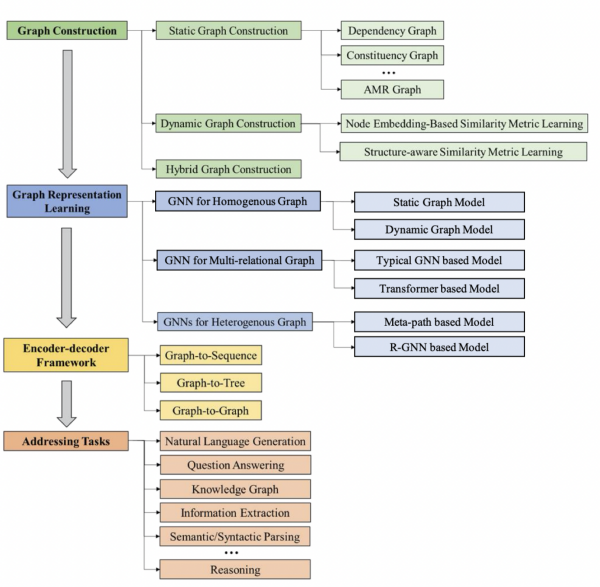

Figure 3: Graph Neural Networks for NLP

question. Figure 1 shows the structure and process of this task.Our work compares two different static graph construction methods, i.e., Dependency Graphs and Abstract Meaning Representation (AMR) Graphs, followed by a discussion on usage of dynamic graphs with GraphVQA framework. Centers around this goal, we also explored a number of variances of this system by using different features, including positional information encoding, bidirectional edges, and more advanced node features from BERT embeddings.

2 Related Work

The representation of natural language has a fundamental influence on the way machines process and understand it. The simplest way of text representation is bag-of-words, but it ignores the information about the position of words or tokens. A better approach is sequence-of-words, it stores the information of word pair co-occurrences in a local context. Alternative approach of natural language representation is using graphs, e.g. dependency graphs, constituency graphs, AMR graphs, IE graphs, lexical networks, and knowledge graphs. The advantages of using graph based representation of natural language over the former ones is that it provides richer relationships among text elements. The reduced complexity of graph methods over vector methods offers a more compressed and efficient concept representation of text. Previous work (Mills and Bourbakis, 2014) has presented a summary of such graph-based methods. It provided an analysis of their component functions and their maturity calculated from information found in its referenced papers. The traditional graph-based methods mostly focus on the structural information of the graphs and do not consider node or edge features, which are also very important for Natural Language Processing (NLP) and Natural Language Understanding (NLU). Therefore, we need a unified graph-based learning framework that offers better representation for graphs structures, node and edge properties. Recently, many studies on extending deep learning approaches for graph-structured data have emerged. A Graph Neural Networks (GNN) are a class of artificial neural networks that can be used to apply deep learning approaches on graph-structured data. Figure 2, shows a general design pipeline for a GNN model (Zhou et al., 2018). There are different types of graph neural networks like Recurrent Graph Neural Networks (RGNNs) (Chen et al., 2019), Convolutional Graph Neural Networks (GCN) (Kipf and Welling, 2016), Graph Autoencoders (GAEs) (Pan et al., 2018), Spatio-temporal Graph Neural Networks (STGNNs) (Peng et al., 2020), Graph Attention Neural Networks (GAT) (Veličković et al., 2017) etc. The paper (Wu et al., 2021b), provides a comprehensive overview of graph neural networks (GNNs) in data mining and machine learning fields.

The prerequisite for GNN's to work is that the input data should be represented as graphs. Since the natural language is not graphically formatted by default, we use static graph construction or dynamic graph construction methods to construct them. The idea of graph construction is to augment the raw text with extracted structural information.

Figure 3, which is taken from paper (Wu et al., 2021a), shows the taxonomy, which systematically organizes GNNs for NLP along three axes: graph construction, graph representation learning, encoder-decoder models, and the applications.

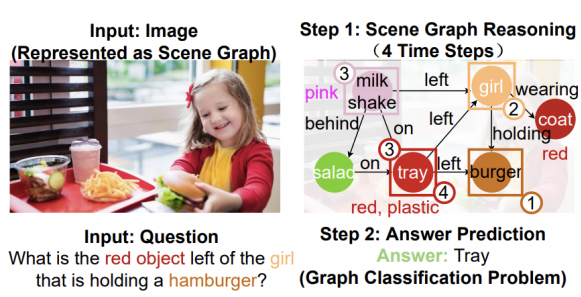

Figure 4: GraphVQA F: Scene Graph

A1. Is the **tray** on top of the **table** black or light brown? light brown
A2. Are the **napkin** and the **cup** the same color? yes
A3. Is the small **table** both oval and wooden? yes
A4. Is there any **fruit** to the left of the **tray** the **cup** is on top of? yes
A5. Are there any **cups** to the left of the **tray** on top of the **table**? no
B1. What is the brown **animal** sitting inside of? **box**
B2. What is the large **container** made of? cardboard
B3. What **animal** is in the **box**? **bear**
B4. Is there a **bag** to the right of the green **door**? no
B5. Is there a **box** inside the plastic **bag**? no

Figure 5: Example of questions from GQA dataset

3 Methods

The GraphVQA framework uses scene graphs instead of raw images as the knowledge base of the question-answering system. We will give a quick overview of this original framework in the following subsections.

3.1 GraphVQA Framework

GraphVQA is a a language-guided graph neural network framework which supports question-answering on scene graphs (Liang et al., 2021). Figure 4 shows an example question: "What is the red object left of the girl that is holding a hamburger". This question can be naturally answered by the following iterations of message passing "hamburger → small girl → red tray". The final state after message passing represents the answer (e.g., tray), and the intermediate states reflect the model's reasoning. Each message passing iteration is accomplished by a graph neural network. (GNN) layer.

3.1.1 Data

It uses the GQA dataset (Hudson and Manning, 2019b), which is a a new dataset for real-world visual reasoning and compositional question answering, seeking to address key shortcomings of previous VQA datasets (Hudson and Manning, 2019a). The dataset has 110K scene graphs, 1.5M questions, and over 1000 different answer tokens. Figure 5, shows some examples of question from the GQA dataset.

3.1.2 Architecture

As we can see from figure 6, the GraphVQA framework consists of the following modules:

- **Question Parsing Module**: This module uses sequence-to-sequence transformer to parse questions into sequence of fixed length instruction vectors.
- **Scene Graph Encoding Module**: It first initialises the nodes and the edges with word embeddings, and then obtains contextualized node features by taking the dot product of node and edge matrices.
- **Graph Reasoning Module**: This module uses the message passing in layer L conditioned on the L^{th} instruction vector. They explored three standard GNNs, i.e., Graph Convolution Networks (GCN), Graph Isomorphism Network (GINE) and Graph Attention Network (GAT). Out of the three, Graph Attention Network (GAT) outperforms the other two for message passing. GraphVQA-GAT outperforms the state-of-the-art model by a large margin (88.43% vs. 94.78%)
- **Answering Module**: The Graph Reasoning Module gives the final states of all graph nodes after **M** iterations of message passing. The final state is then summarized after message passing, following which the answer token will be predicted by this module.

3.2 Our Methods

In the original framework, the architecture is not completely GNN-based because of the question parsing module. Because of the above mentioned benefits of GNN-based language representations, we explored the potential of a GNN-based question parsing module by substituting the original Transformer encoder-decoder structure with a Graph2Seq structure (Xu et al., 2018). Different from Xu et al., 2018, which used GCN as the encoder and RNN as the decoder. We employed more

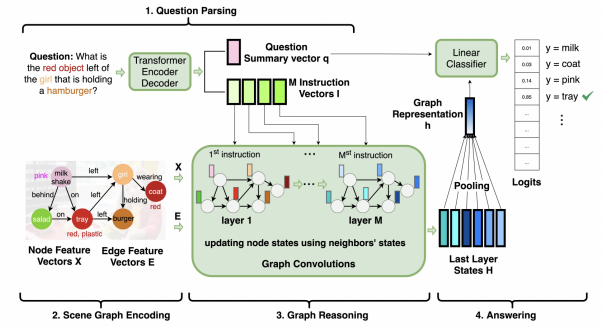

Figure 6: Semantics of the GraphVQA Framework

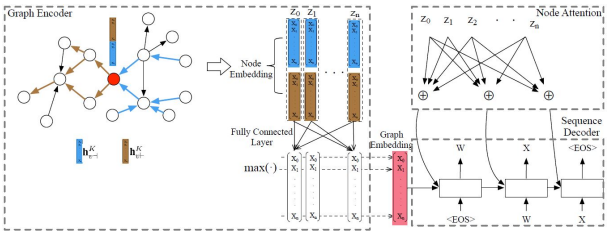

Figure 7: Graph2Seq Question Parsing module

recent technologies by using a graph transformer with unified passing model (Shi et al., 2020) for the encoder and a transformer decoder (See Figure 7 for the structure illustration).

The model takes in the question, and get pass through graph transformer layers and a top-k pooling layer to encode the question into an embedding. This embedding will then be used as the sequence from the last layer of the encoder and fed into the transformer decoder, together with a sequence of reasoning steps that are needed to answer the question provided by GQA dataset.

4 Experiments

We have experimented with two schemes for graph creation methods, i.e, the syntactic dependency tree and the semantic dependency tree.

The experiemnts were conducted with 2 settings for representing the nodes of the question graphs: 1. Using 300 dimension GloVe embeddings for all the nodes; 2. Using fine-tuned BERT representations for all the nodes. The edge embeddings of the encoded questions were always randomly initialized and have 128 dimensions.

To explore the best graph settings, we conducted a series of ablation studies, with *node feature representations*, *Bidirectional edges* and *position information of the nodes* as variables.

We trained all of our models using Adam optimization method, with a learning rate of 10^{-4}, and batch size of 40 due to computing resource limit of the hardware for 100 epochs. GloVe embeddings size is 300, and the BERT embeddings size is of 256 (To control the parameter size, we add on top of the pooling layer a fully connected dense layer with Tanh activation, which performs a down-project to 256 dimensions. Hence, embeddings by this model will only have 256 instead of 768 dimensions.). The instruction vectors have a dimension size of 512. The edge embeddings are always of size 128.

4.1 Data Preprocessing

We parsed the 1.5M questions, to the get the below mentioned static graphs:

4.1.1 Dependency Graphs

A dependency parse tree is a graph where the set of vertices contains the words in the sentence, and each edge connects two words. The intuition behind using dependency parsing is that it finds the relationship between the words in a sentence unlike constituency parsing which only uses the grammar rules, i.e., Context-Free Grammar for parsing. It gives a lot of flexibility even when the order of words (like 'pretty girl' or 'girl pretty') gets changed. Therefore, it can handle morphologically rich languages like English very well. The paper (Barbero and Lombardo, 1995) talks about the suitability of the dependency parsing approach in practical applications of NLP.

We constructed a dependency tree for each question using spaCy `https://spacy.io/api/dependencyparser`. The spaCy parser internally uses transition-based dependency parsing, which jointly learns sentence segmentation and labelled dependency parsing. The parser uses a variant of the non-monotonic arc-eager transition-system described in the paper (Honnibal and Johnson, 2015) with the addition of a "break" transition to perform the sentence segmentation. The parser is also capable to predicting non-projective parses using the (Nivre and Nilsson, 2005) pseudo-projective dependency transformation. Figure 8, shows a dependency tree of a sample question from our dataset.

4.1.2 AMR Graphs

Abstract meaning representation (Banarescu et al., 2013) is a semantic structure formalism. It represents the meaning of a text in a rooted directed graph, where nodes represent basic semantic units

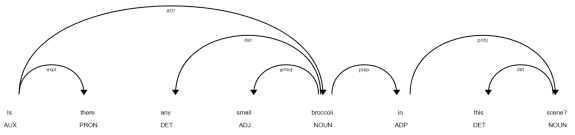

Figure 8: Dependency tree of a sample question from GQA dataset

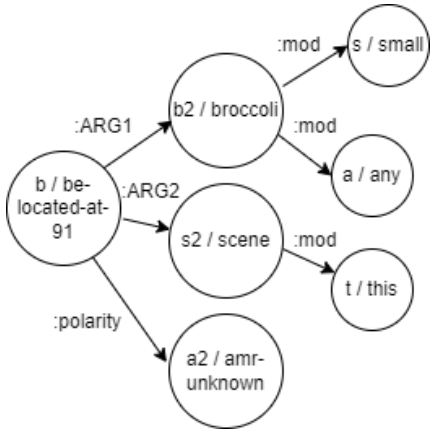

Figure 9: AMR graph of a sample question from GQA dataset

such as entities and predicates, and edges represent their semantic relations, respectively. Refer to the paper (Bai et al., 2022) for more details. The two fundamental NLP tasks solved by AMR graphs are AMR parsing, which means converting text into semantic graphs and AMR-to-text generation.

On our dataset, using AMR parsing technique, an AMR graph for each question was constructed using the python library AMRLib, as an extention to SpaCy. We used **parse_xfm_bart_base** pytorch model and derived the AMR graph in the forward pass using a batch size of 50 questions. Figure 10, shows the model card. The output was converted to triples using penman graph notation library. The computation was done on GPU using parallel processing. Figure 9, shows an AMR graph of a sample question from our dataset.

model_type	BART
num_hidden_layers	6
num_beams	4
encoder_layers	6
transformers_version	4.16.2
vocab_size	50265
max_train_graph_len	512
max_train_sent_len	100

Figure 10: AMR Model Card

4.2 Bidirectional Edges

The graphs that we are handling in this projects should all be trees (acyclic). That means, the node embedding information will be propagated only in a certain direction. To enable bidirectional propagation, we added reverse edges to the graph construction as well. We do that by inventing a new reverse edge for every kind of them, and creating an embedding for it correspondingly. For example, for relation: The $\xleftarrow{\text{det}}$ assult, we add a new edge: The $\xrightarrow{\text{reverse-det}}$ assult. We also added self loop edges for every node, which has a special embedding by itself.

4.3 Positional Information

It has been proven before that the learnable positional encoding helps with the performance (Dwivedi et al., 2021). To explore the benefits of positional feature, we create an extra one-hot position embedding for every node, indicating the absolute position of this word in the sentence. These embeddings will be summed together with the word embeddings and be used as the final input for our system.

4.4 Fine-tuned BERT

BERT has been one of the most successful and popular pre-trained models. Its presence almost always boost the resulting performance of the models. Therefore, we want to bring the advantages in BERT to GNN, and evaluate that how much improvement can BERT help GNN models work. We firstly substitute the original Transformer encoder with a simple BERT encoder and fine-tuned it so the embeddings are adapted to the task, then we used the fine-tuned model to provide representations for all the nodes in our GNN encoder by using *SentenceTransformer* (Reimers and Gurevych, 2019)

5 Results and Discussion

5.1 Ablation Studies

We experimented with a series of ablation studies, by controlling variables include: Embedding choices (GloVe or fine-tuned BERT); With or without bidirectional edges; With or without positional embeddings. To see the full results, please refer to table 1.

The main take-home message is that the combination of BERT, AMR graph scheme with bidirectional edges and positional encoding is the winner.

It also beats the results from the original paper by a small margin of 0.35%. Dependency scheme, on the other hand, does not perform as well as AMR. It makes sense since the AMR graph provides very rich semantic information in contrast to dependency graphs. For example, the root of an AMR graph serves as a elementary representation of the overall focus, making the minimum distance from the root node partially reflect the importance of the corresponding concept in the whole-sentence semantics. Whereas it is not always true in case of dependency graphs. There were also research showing that AMR could be very helpful in question answering tasks (Ngomo, 2018). We can expect an another improve in the future once the accuracy of AMR generation tool progress even more.

Another important information is that the quality of embeddings for the nodes also makes a major difference generally. This point will be further analysed in the following subsection.

5.2 Results across Question Categories

In Table 2 we present the model's performance across question categories. Among all types of questions, verify, choose and compare are binary; query and logical are open questions. As we can see, the model improves the performance on binary questions stably. We believe that there are two main reasons for that. Firstly, there are are disproportionately less question in types of *compare* (589 questions) and *choose* (1803 questions) in the test set, making even a few more right prediction can boost the accuracy higher comparing to other question types (for example, query type questions, which have 6805 entries in the test set). Secondly, if we inspect the questions in each type, we can find out that the binary questions are usually longer and more complicated than the open ones, because those questions usually have a description stating the two options after the core question. It would require the question parsing module to understand the meaning of the choices, and the relation between the choices and the core question. Take a *choose* type question as an example: *What is the animal that is setting next to the dog (core question), a cat or a rabbit?* (the options). If it were a open *query* type of question, it would simply be: *What is the animal that is setting next to the dog?*. Because of this, we can infer that our GNN system does a better job at capturing the semantics of long sentences. The results are also expected and consistent with current studies (Xu et al., 2018), because parsing a sequence into a graph structure could make the message passing more efficient and the additional semantic or syntactic information in graph structure could help with the question encoding process.

5.3 Bottleneck Analysis

Although our methods improved the performance based on the original paper, the margin is not extraordinarily big. A possible reason is that question encoding is not the bottleneck of the system compare to scene graph encoding and reasoning module. This problem is common in multi-modal system and it is also consistent with the ablation study that conducted by the original paper: With only questions, the system achieved 41.07% overall accuracy, whereas with only scene graphs, the accuracy is only 19.63%.

Another possible bottleneck is the performance of the AMR tree generator. Although we are using a SOTA generator with 83.7 match score, there is still room for improvement. But what worth noticing is, the match score was calculated on much complicated sources of corpora, including broadcast conversations, newswire, weblogs, web discussion forums, fiction and web text. The questions in the dataset usually have simpler structure and the match score on this dataset should be considerably higher.

6 Conclusion

GraphVQA is a framework powered by state-of-the-art neural networks, which supports question-answering on scene graphs. In the question parsing module of GraphVQA, we replaced the Transformer encoder-decoder structure with a Graph2Seq structure for representing the questions. In our work we compared the results using two static graph creation methods, i.e., the Dependency graphs and the AMR graphs with different node embedding creation methods (Glove and BERT) and with or without Bidirectional Edges and Position Embeddings. We show the successful implementation and comparison of 12 model variants. The GNN with BERT embeddings using AMR graphs having both Bidirectional Edges and Position Embeddings outperformed all and also the original framework. In the future work section, we discuss about the dynamic graphs, their advantages over static graphs and also suggest a framework for easy

Node Embed.	Scheme	Bidirectional Edges	Position Embed.	Acc
GloVe	Dependency	-	-	93.22
GloVe	AMR	-	-	94.90
BERT	Dependency	-	-	94.31
BERT	AMR	-	-	94.94
GloVe	Dependency	yes	-	94.19
GloVe	AMR	yes	-	94.92
BERT	Dependency	yes	-	94.71
BERT	AMR	yes	-	95.03
GloVe	Dependency	yes	yes	94.64
GloVe	AMR	yes	yes	95.10
BERT	Dependency	yes	yes	94.74
BERT	**AMR**	**yes**	**yes**	**95.13**

Table 1: Ablation study results. *Bidir* refers to bidirectional edges; *Pos* refers to positional embeddings

Model	Verify	Choose	Compare	Query	Logical
Our Best	97.47	94.56	93.88	93.96	99.02
Original	96.98	93.57	92.96	93.37	98.79
Margin	+0.49	+0.99	+0.92	+0.59	+ 0.23

Table 2: The performance of our best model across all types of questions.And the comparison to the original model

implementation and integration of the same with our current GNN-based system.

7 Future Work

Due to time and resource limitations, our experiments only focused on static graphs. In the future work, we expect to use dynamic graph for the both question parsing and scene graph reasoning module.

A dynamic graph system has properties of changing graph rather than recomputation from scratch after each change. They dynamically learn the graph structure, i.e. the weighted adjacency matrix. A fully dynamic graph has both insertion and deletion of edges, whereas a partially dynamic algorithm only allows insertions (Eppstein et al., 1997). The advantage of using dynamic graphs over static graphs is that, the graph construction module can be jointly optimised with the GNN's, refer to figure 11, taken from https://graph-neural-networks.github.io/static/file/chapter14.pdf.

There are various types of dynamic graphs and creation methods in the literature, some of them are Graph Similarity Metric Learning, Graph Sparsification Techniques, Combining Intrinsic and Implicit Graph Structures, Node Embedding Based

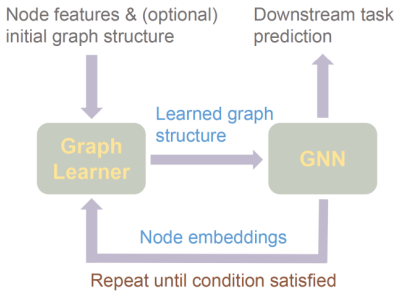

Figure 11: Iterative and Joint learning

Graphs, Structure-aware Graphs etc.

The great success of GNNs relies on the quality and availability of graph-structured data which can either be noisy or unavailable or the intrinsic graph topology might merely represent physical connections (e.g the chemical bonds in molecule), and fail to capture abstract or implicit relationships among vertices. Keeping the challenges in mind, we suggest a graph sparsification technique for dynamic graph creation that aims to discover useful graph structures from data, to help solve the above issue. The graph sparsification technique approximates the large input graph with a sparse subgraph to enable efficient computation, and at the same time preserve certain properties. To do so, we recommend to use the NeuralSparse framework's super-

vised graph sparsification technique, which could seamlessly connect with existing graph neural networks for more robust performance (Zheng et al., 2020). The architecture of NeuralSparse is called PTDNET, and it has two components, the denoising networks and the GNNs. The denoising network is multi-layer network that samples a sub-graph from the learned distribution of edges. The advantages of using PTDNET is that, it is compatible with most existing GNNs, such as GCN, GraphSage, GAT, GIN, etc. As a piece of future work, we can replace the dependency trees and AMR graphs with the this technique of dynamic graph creation and jointly optimise it along with the Graph Attention Networks used in our system and evaluate the overall performance of GraphVQA.

References

Xuefeng Bai, Yulong Chen, and Yue Zhang. 2022. Graph pre-training for AMR parsing and generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6001–6015, Dublin, Ireland. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Cristina Barbero and Vincenzo Lombardo. 1995. Dependency graphs in natural language processing. pages 115–126.

Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alexander G. Hauptmann. 2021. Scene graphs: A survey of generations and applications. *ArXiv*, abs/2104.01111.

Yu Chen, Lingfei Wu, and Mohammed J Zaki. 2019. Graphflow: Exploiting conversation flow with graph neural networks for conversational machine comprehension. *arXiv preprint arXiv:1908.00059*.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2021. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*.

David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. 1997. Sparsification—a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696.

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.

Ronghang Hu, Anna Rohrbach, Trevor Darrell, and Kate Saenko. 2019. Language-conditioned graph networks for relational reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10294–10303.

Drew Hudson and Christopher Manning. 2019a. Gqa: A new dataset for real-world visual reasoning and compositional question answering. pages 6693–6702.

Drew A Hudson and Christopher D Manning. 2019b. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Linjie Li, Zhe Gan, Yu Cheng, and Jingjing Liu. 2019. Relation-aware graph attention network for visual question answering. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10313–10322.

Weixin Liang, Yanhao Jiang, and Zixuan Liu. 2021. GraghVQA: Language-guided graph neural networks for graph-based visual question answering. In *Proceedings of the Third Workshop on Multimodal Artificial Intelligence*, pages 79–86, Mexico City, Mexico. Association for Computational Linguistics.

Michael T. Mills and Nikolaos G. Bourbakis. 2014. Graph-based methods for natural language processing and understanding—a survey and analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(1):59–71.

Ngonga Ngomo. 2018. 9th challenge on question answering over linked data (qald-9). *language*, 7(1):58–64.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics.

Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407*.

Hao Peng, Hongfei Wang, Bowen Du, Md Zakirul Alam Bhuiyan, Hongyuan Ma, Jianwei Liu, Lihong Wang, Zeyu Yang, Linfeng Du, Senzhang Wang, et al. 2020.

Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting. *Information Sciences*, 521:277–290.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. 2021a. Graph neural networks for natural language processing: A survey.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021b. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. 2018. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.

Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Robust graph representation learning via neural sparsification. In *International Conference on Machine Learning*, pages 11458–11468. PMLR.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications.