



**Czech  
Technical University  
in Prague**

**F3**

Faculty of Electrical Engineering  
Department of Computer Science

## **Graph Neural Networks for Long Input NLP Models**

Master Thesis of  
**Václav Hlaváč**

Supervisor: **Ing. Jan Drchal, Ph.D.**  
Field of study: **Open Informatics**  
Subfield: **Data Science**  
**January 2023**



## I. Personal and study details

Student's name: **Hlavá Václav** Personal ID number: **474528**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Computer Science**  
Study program: **Open Informatics**  
Specialisation: **Data Science**

## II. Master's thesis details

Master's thesis title in English:

**Graph Neural Networks for Long Input NLP Models**

Master's thesis title in Czech:

**Metody grafových neuronových sítí pro zpracování dlouhých vstup NLP model**

Guidelines:

Experiment with Graph Neural Networks (GNN) methods aimed at processing long inputs of NLP models (> 1000 tokens).

- 1) Explore the state-of-the-art GNN methods and methods as well as modern neural text classifier architectures.
- 2) Select or develop an appropriate dataset(s) for textual classification.
- 3) Implement one or more GNN-based classifiers and assess their performance compared to the baseline non-GNN approach (such as the BERT classifier) - this would be possible for relatively short input texts, only.
- 4) Evaluate the best classifier architecture on long-input Czech fact-checking datasets supplied by the supervisor.
- 5) Publish the code and data as open-source.

Bibliography / sources:

- [1] Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32.1 (2020): 4-24.
- [2] Malekzadeh, Masoud, et al. "Review of graph neural network in text classification." 2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). IEEE, 2021.
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 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), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- [4] Drchal, Jan, et al. "CsFEVER and CTKFacts: Czech Datasets for Fact Verification." arXiv preprint arXiv:2201.11115 (2022).

Name and workplace of master's thesis supervisor:

**Ing. Jan Drchal, Ph.D. Artificial Intelligence Center FEE**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **14.09.2022** Deadline for master's thesis submission: **10.01.2023**

Assignment valid until: **19.02.2024**

Ing. Jan Drchal, Ph.D.  
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

### III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature

## Acknowledgements

I would like to thank my supervisor Ing. Jan Drchal, Ph.D for providing helpful guidance throughout the whole course of writing this thesis. I would also like to thank my family for the constant support that they show me.

## Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used. I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague, 06. January 2023

## Abstract

The usefulness and the impact of natural language processing methods is becoming ever more relevant. The introduction of the Transformer allowed to easily tackle many tasks, such as question answering and document summarization, using a single architecture. Nevertheless, its quadratic computational complexity and memory requirements make it infeasible to use for processing long input sequences. A majority of the pre-trained models based on the Transformer architecture has a restriction on the length of the input, and usually accepts sequences of length up to 512 tokens. This thesis constructs a classifier of long input sequences by combining the already pre-trained Transformer-based models for short inputs with graph neural networks, a class of deep learning methods for processing graph data. The resulting model is able to process inputs as long as 13,926 tokens. The proposed model is tested on three distinct datasets and its performance is reported and discussed. Furthermore, its possible drawbacks are identified and future work directions for the use of graph neural networks for long input classification are suggested.

**Keywords:** natural language processing, transformer, graph neural networks, classification, long input processing

**Supervisor:** Ing. Jan Drchal, Ph.D.

## Abstrakt

Užitečnost a dopady metod zpracování přirozeného jazyka stále nabývají na důležitosti. Představení Transformerů dovolilo snadné řešení mnoha úloh, jako například zodpovídání otázek nebo sumarizace dokumentů, pomocí jednotné architektury. Jejich kvadratická výpočetní složitost a pametové nároky ale zapřičiňují problematické zpracování dlouhých vstupních sekvencí. Většina předtrénovaných modelů založených na Transformer architektuře zavádí omezení pro délku vstupu a běžně přijímá vstupy délky až 512 tokenů. Tato diplomová práce konstruuje klasifikační model dlouhých vstupních sekvencí kombinací již předtrénovaných jazykových modelů založených na Transformer architektuře určených pro zpracování kratších vstupů a grafových neuronových sítích, třídě metod hlubokého učení pro zpracování grafových dat. Výsledný model je schopen zpracovat například vstupy obsahující 13926 tokenů. Navrhovaný model je testován na třech různých datových sadách a jeho klasifikační schopnosti jsou představeny a diskutovány. Možné nedostatky představeného modelu jsou následně identifikovány a další možné postupy pro využití grafových neuronových sítích pro klasifikaci dlouhých vstupů jsou navrženy.

**Klíčová slova:** zpracování přirozeného jazyka, transformer, grafové neuronové sítě, klasifikace, zpracování dlouhého vstupu

# Contents

<b>1 Introduction</b>	<b>1</b>		
<b>2 Natural Language Processing</b>	<b>3</b>		
2.1 The Transformer	3		
2.1.1 The Attention Mechanism	5		
2.2 Bidirectional Encoder Representations from Transformers	6		
<b>3 Graph Neural Networks</b>	<b>9</b>		
3.1 Classification of Graph Neural Networks	9		
3.1.1 Recurrent Graph Neural Networks	9		
3.1.2 Convolutional Graph Neural Networks	10		
3.2 Message Passing Neural Networks	11		
3.3 Graph Convolution Networks	11		
3.4 Graph Attention Networks	12		
3.4.1 Modification of the Attention Mechanism	13		
3.5 Application of Graph Neural Networks	13		
3.5.1 Application examples	14		
<b>4 Problem Statement</b>	<b>17</b>		
4.1 Related Literature	17		
4.1.1 Longformer	17		
4.1.2 Big Bird	18		
4.1.3 Performer	19		
<b>5 Proposed Solution</b>	<b>21</b>		
<b>6 Datasets</b>	<b>25</b>		
6.1 MultiSource	25		
6.1.1 Dataset Generation Process	25		
6.1.2 Document Sampling	26		
6.2 Hyperpartisan News Detection	28		
6.3 CTKFacts	30		
<b>7 Experiments</b>	<b>33</b>		
7.1 Experimental Setup	33		
7.1.1 Baselines	34		
7.1.2 Implementation Details	35		
7.2 MultiSource	35		
7.2.1 Hyperparameters selection	35		
7.2.2 Task Difficulty Analysis	37		
7.2.3 Importance of Graph Neural Networks	38		
7.3 Hyperpartisan News Detection	39		
7.3.1 Language Models Comparison	40		
7.3.2 Baseline Comparison	41		
7.4 CTKFacts	41		
7.4.1 Initial Experiments	42		
7.4.2 Changing the Amount of Available Evidence	44		
7.4.3 Embedding the Importance of the Evidence	45		
<b>8 Discussion</b>	<b>47</b>		
<b>9 Conclusion</b>	<b>49</b>		
<b>Bibliography</b>	<b>51</b>		

# Figures

2.1 Illustration depicting the architecture of the Transformer. Left side of the illustration shows the encoder, whose output is then fed to the decoder, shown on the right side. Reprinted from [Vaswani et al., 2017]. . . . .	4
2.2 Figure illustrating the attention coefficients computed between different pairs of tokens in a model for machine translation. The tokens on the x-axis are from the original input sequence, and the y-axis shows the tokens of its generated French translation. Reprinted from [Bahdanau et al., 2014] . . . . .	5
3.1 Figure showing the application of multi-head attention in GAT. The different styles of arrows signify computations of different heads. Reprinted from [Veličković et al., 2017]. . . . .	13
4.1 Comparison of the full attention mechanism pattern and the attention patterns introduced in the Longformer [Beltagy et al., 2020] paper. Reprinted from [Beltagy et al., 2020]. . . . .	18
4.2 Comparison of the different attention patterns that make up the Big Bird’s sparse self-attention mechanism. Reprinted from [Zaheer et al., 2020]. . . . .	18
5.1 Figure showing the two proposed graph topologies. Section a) of the figure shows a <i>complete</i> graph with six nodes, and section b) shows the <i>local + global</i> topology configuration with the additional latent node, which is shown in orange. Self-loops are not shown for clarity. . . . .	22
5.2 An overview diagram of the proposed model. The input document is first split into multiple parts. An embedding is created for every part using a Transformer-based language model. These embeddings are then used as features of the nodes in the constructed graph, which connects different parts of the input. Finally, a graph neural network composed of multiple GNN layers is used to obtain the final classification of the input document. . . . .	23
6.1 A small MultiSource example generated using the similar sampling. It is composed of four sentences, where two come from a single identical source, and the other two sentences originate from two different sources. . . . .	27
6.2 A small MultiSource example. The sentences originating from different sources were sampled from documents obtained using random sampling, without regards to the ‘original’ source document. . . . .	28
6.3 The cumulative distribution function of the token counts of the Hyperpartisan News Detection dataset examples. Most of the examples contain more than 512 tokens. The median length of an example is 863 tokens, and the dataset contains examples as long as 13,926 tokens. . . . .	29
6.4 An example of a hyperpartisan news article from the Hyperpartisan News Detection dataset. . . . .	29
6.5 An example of a claim from the training split of the CTKFacts dataset, along with its evidence. The claim is supported by the information contained in the knowledge base. The example is reprinted from [Drchal et al., 2022]. . . . .	30



6.6 The automated fact checking pipeline. The model proposed in this thesis performs the natural language inference step. The classification label can be one of <b>SUPPORTS</b> , <b>REFUTES</b> , and <b>NOT ENOUGH INFORMATION</b> . . . . .	31
7.1 Confusion matrix for the test set of CTKFacts. The predictions were obtained using our proposed model with FERNET-C5 as the encoder, <i>local + global</i> topology, and 2 GATv2 layers with 4 heads. The node embedding size was set to 256. . . . .	44
7.2 An example of 128-dimensional positional encodings, generated for the first 50 positions. . . . .	45

## Tables

6.1 The class distributions for different splits of the Hyperpartisan News Detection dataset. . . . .	28
6.2 The class distributions for different splits of the CTKFacts dataset. . . . .	31
7.1 Configuration of the MultiSource-Mix-40k dataset splits. The individual entries indicate the number of examples. . . . .	36
7.2 The test set accuracies of our model with different choices for the graph neural network hyperparameters on the MultiSource-Mix-40k dataset. The baseline test accuracy, obtained using the BERT language model, was 89.55%. . . . .	36
7.3 The influence of the node embedding's size on the classifier's performance. Results shown are the test accuracies. . . . .	37
7.4 Results of an experiment showing the influence of the sampling strategy on the difficulty of the MultiSource task. Along with the test accuracy of our model, we also show the test accuracy of the BERT baseline. . . . .	37
7.5 Results of the experiment inspecting the influence of the ratio of sentences from a single source and sentences from different sources. Shows comparison between different topologies and the effect of using a sliding window over sentences. . . . .	38
7.6 The results of the graph neural networks ablation experiment. The best results for each of the window sizes are shown in bold. . . . .	39
7.7 The F1 validation scores for different choices of the GNN settings for the <i>complete</i> topology. . . . .	40
7.8 The F1 validation scores for different choices of the GNN settings for the <i>local + global</i> topology. . . . .	40

7.9 Comparison of F1 scores on the validation split when the BERT model is swapped for RoBERTa. Interestingly enough, using RoBERTa does not yield better results. . . . .	40
7.10 The resulting F1 test scores. Our proposed model with the <i>local + global</i> topology outperforms the baseline BERT model and matches the Longformer in the F1 metric. . .	41
7.11 Comparison of validation F1 micro scores when using the cross-lingual language models, mBERT and XLM-RoBERTa @ SQuAD2, as encoders along with different configurations of the GNNs with <i>complete</i> topology. . . . .	42
7.12 Comparison of validation F1 micro scores when using the Czech monolingual language models, FERNET-C5 and RobeCzech, as encoders along with different configurations of the GNNs with <i>complete</i> topology. . . . .	43
7.13 Comparison of CTKFacts validation F1 micro scores for different configurations of GNNs with <i>local + global</i> topology and different language models. . . . .	43
7.14 Comparison of our best performing model configuration in the <i>complete</i> and <i>local + global</i> topology settings and the FERNET-C5 baseline. . . . .	44
7.15 The resulting CTKFacts test set performances of our classifier for different numbers of provided evidence. . . . .	45
7.16 The influence of embedding the evidence importance using positional encoding. . . . .	46



# Chapter 1

## Introduction

Natural Language Processing is a field that enjoys great attention. It offers a lot of opportunities for practical use and automation, ranging from document classification to machine translation and chatbots. It underwent a significant shift with the introduction of Transformers. The Transformer architecture, despite being initially designed for neural machine translation, is a universal approximator of sequence to sequence functions. It is useful for other tasks as well, such as text generation, classification, question answering, or document summarization. As opposed to recurrent neural networks, it is not necessary for Transformers to process the input tokens sequentially, but instead are able to process them in parallel. This allows the models to process large quantities of data, which helps them learn complex tasks. In spite of their popularity, most language models based on the Transformer architecture have a restriction on the length of the input sequence. That is due to their quadratic computational complexity and memory requirements in terms of the input sequence length.

This diploma thesis aims to construct a classifier that is able to process input sequences of lengths that exceed the limitations of the Transformer-based models. We will attempt to do so using graph neural networks, a class of deep learning methods for processing graph data. At the same time, we will reuse the *already pre-trained* language models, despite their limitations. This will also give us the ability to process long inputs that would in some cases require us to perform resource-demanding pre-training of Transformer models for long inputs from scratch.

The remainder of this thesis is structured as follows.

Chapter 2 introduces the field of natural language processing. It describes how the Transformer architecture came about, the motivations behind it, and its initial use cases. It is then described more in detail, along with its core technique, the self-attention mechanism. Finally, we introduce BERT, the Transformer-based language model for generating sequence representations.

We lay out an overview of graph neural networks in Chapter 3. Different categories of graph neural networks are introduced and described. We also present a number of specific graph neural network methods, the types of tasks that graph neural networks can solve, and examples of their applications.

In Chapter 4 we state the problem with Transformer-based models, which we will attempt to address, and quickly go over a few methods that were designed to tackle it.

Chapter 5 introduces our proposed model and describes its inner workings. We describe the way we utilize graph neural networks to process long input sequences.

We go over three classification tasks and their respective datasets used to assess the model's performance in Chapter 6. We describe each one of them and give examples of what their data looks like. One of these three is also a task that we introduce as a part of this work and that we designed so that it is possible to control its difficulty.

In Chapter 7 we carry out experiments using the previously introduced datasets and compare the performance of our model to the baseline methods.

We discuss our proposed model, go over some of its drawbacks, and suggest possible future work in Chapter 8. Finally, we conclude the thesis in Chapter 9.

## Chapter 2

# Natural Language Processing

The aim of **natural language processing (NLP)** is to understand the information conveyed using natural language and subsequently use this understanding to solve various tasks. For example, the tasks that NLP tackles include document and sentiment classification, named entity recognition, machine translation, and text summarization.

Some of these tasks, such as machine translation and text summarization, require the model to generate a new text based on the one it was given. Initial approaches to translating a text from one language to another mostly involved rule-based systems and statistical methods [Brown et al., 1990].

The subsequent neural network based approaches for machine translation usually follow an *encoder-decoder* framework. An encoder reads the input sequence and encodes it into a vector of a fixed dimension. This vector is passed to the decoder, which then utilizes this encoding to generate individual output sequence tokens.

Most of the time, the encoder and decoder modules were recurrent neural networks (RNNs), such as the Long short-term memory networks [Hochreiter and Schmidhuber, 1997, Sutskever et al., 2014]. These neural networks sequentially read an input sequence while maintaining a hidden state vector. The final hidden state vector is then used as the encoding of the sequence.

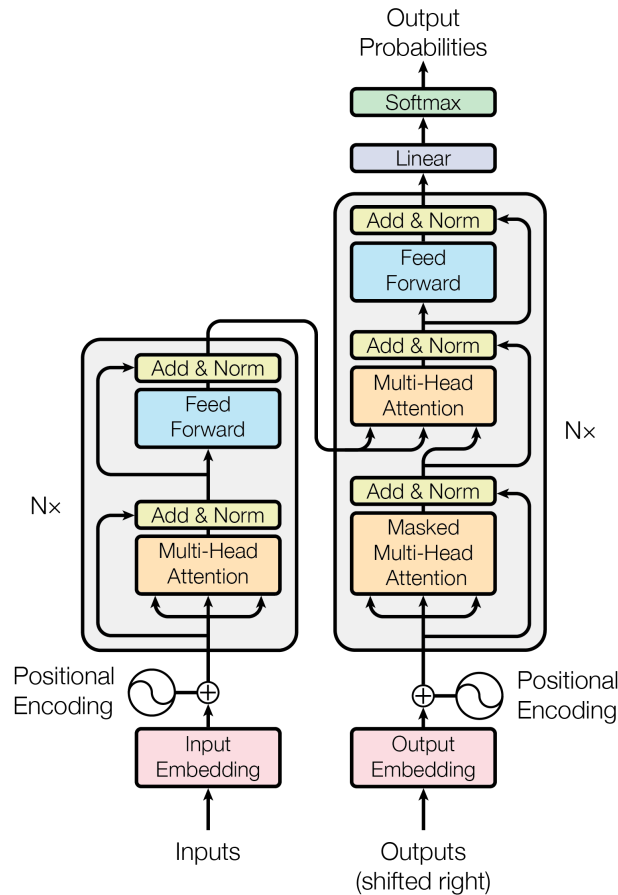
[Bahdanau et al., 2014] recognized that encoding the input sequence into a single vector of a fixed dimension is likely a bottleneck that restricts the ability of the model to extract the necessary information for the generation of a proper translation. Therefore, they proposed the usage of a technique called **the attention mechanism**, which utilizes individual hidden states generated during the recurrent computation. These are used when generating the output sequence, where every new output token being generated can choose which of the input tokens' hidden states it should pay attention to and assigns it a score, which signifies its importance.

## 2.1 The Transformer

The renowned paper by [Vaswani et al., 2017], called 'Attention Is All You Need', has shown that the attention mechanism is very powerful on its own, without the involvement of RNNs. It introduced a natural language processing model called the Transformer, whose central components rely solely on this mechanism.

More specifically, the Transformer is based on the self-attention mechanism, which is essentially equivalent to the original attention mechanism. The main difference is that the tokens of the input sequence pay attention to other tokens of that same sequence, and as a result, the self-attention mechanism computes a representation of the input sequence.

The decision to use only the attention mechanism allows the model to disregard any



**Figure 2.1:** Illustration depicting the architecture of the Transformer. Left side of the illustration shows the encoder, whose output is then fed to the decoder, shown on the right side. Reprinted from [Vaswani et al., 2017].

recurrence computations, which were used prominently in the field of natural language processing. The usage of recurrent models turned out to be a hurdle in modeling long-range dependencies contained within input sequences. The Transformer does not have this limitation, since the attention mechanism allows every token to attend directly to any other token in the sequence. Furthermore, recurrence computations do not lend themselves well to computation optimizations because of their sequential nature, whereas the partial computations of the attention mechanism can be done for every token independently.

The flexibility of Transformer-based models allowed them to tackle tasks such as question answering, natural language inference, and text summarization.

The original Transformer follows the encoder-decoder architecture, where the input tokens are transformed into context-aware embeddings by the encoder stack, which are then utilized by the decoder stack to generate the output sequence. Its architecture is illustrated in Figure 2.1.

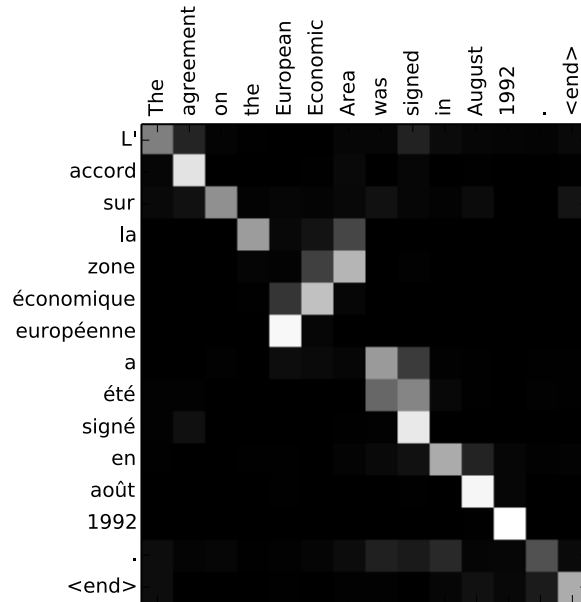
The Transformer model and its descendants are one of, if not the most, popular and utilized NLP models at the time of writing this thesis.

Apart from being used in NLP, the Transformer architecture has also been modified so that it can be used for example in computer vision for image classification or object detection [Chen et al., 2020, Carion et al., 2020, Liu et al., 2021] or to process audio and perform speech recognition [Gulati et al., 2020, Chen et al., 2021, Radford et al., 2022].

### 2.1.1 The Attention Mechanism

This section takes a closer look at the inner workings of the attention mechanism used in the Transformer-based models, which the authors call the scaled dot-product attention.

The attention mechanism receives three different vectors for each of the tokens of the processed sequence. These three vectors are called the **query**  $\mathbf{q}$ , the **key**  $\mathbf{k}$ , and the **value**  $\mathbf{v}$ .



**Figure 2.2:** Figure illustrating the attention coefficients computed between different pairs of tokens in a model for machine translation. The tokens on the x-axis are from the original input sequence, and the y-axis shows the tokens of its generated French translation. Reprinted from [Bahdanau et al., 2014]

Then, for every token  $i$ , the attention mechanism determines its attention score for every sequence token  $j$  by computing the dot-product of query vector  $\mathbf{q}_i$  and key vector  $\mathbf{k}_j$  and applying the softmax function to all of these query-key dot-products. The output of the attention mechanism for the token  $i$  is then a linear combination of the value vectors weighted by their respective attention scores.

These operations can be easily expressed in matrix form, given that we pack the query, key, and value vectors in matrices  $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d_k}$ ,  $\mathbf{V} \in \mathbb{R}^{n \times d_v}$  respectively.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad (2.1)$$

The authors of the Transformer architecture further introduced a novel technique called **multi-head attention**. This technique allows the model to compute the attention mechanism for different transformations of the input vectors. The outputs are then concatenated and projected to form the final output. This gives the model the ability to capture different types of information contained within the sequence and combine them together.

The multi-head attention is computed as follows

$$\begin{aligned} \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)\mathbf{W}^O \\ \text{head}_i &= \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right) \end{aligned} \quad (2.2)$$

where  $\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ , and  $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$  are the projection matrices.

The encoder stack of the Transformer is made up of several encoder modules. These perform the multi-head attention mechanism and apply an additional feed-forward neural network to each of its outputs. The input vectors supplied to the first of these encoders consist of token embeddings, which are learned during the training of the model.

One can see that the computation is invariant to the permutation of the tokens. Since the order of the tokens can be an important piece of information, the initial token embeddings are also enhanced with positional encodings, which are constructed using a combination of the sine and cosine functions.

One of the downsides of the attention mechanism is that it has a  $\Theta(n^2)$  computational and memory complexity, where  $n$  is the number of input tokens. That is because an attention score is computed between every pair of tokens.

There is no dependency between the computations of the attention coefficients for the individual source tokens. Thanks to that fact, they can be expressed in a matrix form and computed simultaneously, as shown in equation 2.1. Nevertheless, the quadratic complexity makes it difficult to process longer inputs.

## ■ 2.2 Bidirectional Encoder Representations from Transformers

In spite of the Transformer being originally introduced as a model for sequence-to-sequence modeling, it has been found that the embeddings generated by the Transformer's encoder stack can form very useful representations of the input sequences.

This is something that one of the most popular modifications of the Transformer model, called BERT [Devlin et al., 2018], makes use of. BERT, or the Bidirectional Encoder Representations from Transformers, expands on the idea of learning useful input embeddings.

From the perspective of the model's architecture, BERT's construction is identical to that of the encoder stack of the Transformer. However, as opposed to unidirectional language models, it allows the attention mechanism to attend to both the previous and the following tokens in the sequence, thus creating a bidirectional language model that is better at capturing the context of the entire input.

The model is pre-trained using the *Masked Language Modelling (MLM)* and the *Next Sentence Prediction (NSP)* tasks, which require an understanding of both the left and the right context.

- **Masked Language Modelling.** A token-level prediction task. A percentage of the tokens is replaced by special [MASK] tokens, and the objective is to predict the original tokens based on the context given by the other tokens in the input sequence.
- **Next Sentence Prediction.** A sequence-level prediction task. Two sentences are given to the model, with a special [SEP] token used as a separator between them. The task is then to predict whether these two sentences follow each other in the original document.

The pre-trained model can then be quite easily *fine-tuned* for a specific downstream task, simply by adding an appropriate layer on top of the BERT's output and training this extended model in an end-to-end fashion. Fine-tuned BERT models have shown the ability to perform well on various downstream tasks, such as question answering, sentiment classification, or natural language inference.







## Chapter 3

# Graph Neural Networks

Information can be represented in various ways and take on different forms. One of the forms that often occurs naturally is that of a graph structure. Formally, a graph is a pair  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V}$  is a set of vertices (or nodes) and  $\mathcal{E}$  is a set of edges connecting these vertices. For directed graphs, the edges are represented as ordered vertex pairs  $(v_i, v_j)$ , where  $v_i, v_j \in \mathcal{V}$  are the source and the target nodes, respectively. For undirected graphs, the edges are symmetrical and therefore represented as an unordered pair of vertices  $\{v_i, v_j\}$ .

Graphs usually model relationships between different objects. They can be found in many places. Take for example a social network, where each vertex would represent a person, and an edge between two vertices might indicate that there exists a friendship between the two individuals. Another example might be that of a molecule structure, which can be represented using a graph whose nodes represent the molecule's atoms and edges represent the bonds that connect them. Many interesting domains can be transformed so that the information they contain can be conveyed using graph structures. The individual nodes and edges of a graph can be enhanced with their respective feature vectors, describing their properties.

Processing graph data using deep learning methods has its difficulties. One needs to account for different aspects of the graph data, such as variable-sized node neighbourhoods, dependencies between the nodes of the graph, and the requirement for the model to be invariant or equivariant to the permutations of the input.

Generally, the neural-based methods that accept and are able to process graph data are called **Graph Neural Networks (GNNs)**.

This chapter introduces the main categories of graph neural networks and briefly presents some of their specific implementations.

### 3.1 Classification of Graph Neural Networks

[Wu et al., 2020] introduced a classification of different approaches of handling graph data using neural networks. These are namely Recurrent GNNs, Convolutional GNNs, graph autoencoders and spatio-temporal GNNs. In this section we introduce more closely the first two of these categories.

#### 3.1.1 Recurrent Graph Neural Networks

Methods that classify as **recurrent graph neural networks (RecGNNs)** were one of the first proposed approaches in the field of GNNs. They update node representations by exchanging information among nodes of a neighbourhood, until a convergence criterium on

the states of the nodes is satisfied. The representations are transformed using a function  $f$  whose parameters are fixed throughout the whole computation and are shared among the nodes. It is required for this function to represent a contraction mapping, so that the states of the nodes converge. This approach can be further modified so that the node states are updated only for a fixed number of time steps, which allows to disregard the contraction mapping restriction on  $f$ .

### 3.1.2 Convolutional Graph Neural Networks

The **convolutional graph neural networks (ConvGNNs)**, also work by propagating information through node neighbourhoods. However, the propagation takes place only for a fixed number of steps, and neural network layers with different weights can be used for each of the steps.

This allows them to work around computationally demanding recurrent updates. However, in doing so, they effectively restrict the amount of information each nodes has access to and can account for in its final representation. By performing the propagation for  $k$  steps, any node has access to information only from the nodes in its  $k$ -neighbourhood.

The ConvGNNs can be further classified into *spectral-based* and *spatial-based* approaches.

#### Spectral-based ConvGNNs

The spectral-based methods perform the convolution operation in the spectral domain of the graph's Laplacian matrix. They have a strong mathematical foundation in signal processing [Sandryhaila and Moura, 2013, Shuman et al., 2013].

Formally, a graph's normalized Laplacian matrix is defined as  $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ , where  $\mathbf{A}$  is the graph's adjacency matrix and  $\mathbf{D}$  is a diagonal matrix of its node degrees. Thanks to being positive semidefinite, the Laplacian matrix has an eigen-decomposition  $\mathbf{L} = \mathbf{U}^T\mathbf{\Lambda}\mathbf{U}$ , where  $\mathbf{U}$  is a matrix of eigenvectors and  $\mathbf{\Lambda}$  is a diagonal matrix of their respective eigenvalues. The graph Fourier transform for a node's feature vector  $\mathbf{x} \in \mathbb{R}^m$  and its reverse are

$$\mathcal{F}(\mathbf{x}) = \hat{\mathbf{x}} = \mathbf{U}^T \mathbf{x} \mathcal{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U} \hat{\mathbf{x}} \quad (3.1)$$

After projecting the node's features into coordinates of the space spanned by the eigenvectors of  $\mathbf{L}$ , a filter  $\mathbf{g}_\theta$  is applied to perform the convolution.

$$\mathbf{x} * \mathbf{g}_\theta = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g}_\theta)) = \mathbf{U} \left( \mathbf{U}^T \mathbf{x} \odot \mathbf{U}^T \mathbf{g}_\theta \right) \quad (3.2)$$

The differences between individual spectral-based ConvGNNs lie in the choice of the filter  $\mathbf{g}_\theta$ , which often belong to a family of parametrized learnable functions.

Since the computation is based around the graph's Laplacian matrix, the learned filter  $\mathbf{g}_\theta$  will be inevitably influenced by the structure of the graph. This unfortunately means that once trained, the spectral-based methods are not able to generalize to graphs of different sizes and structures. Nevertheless, this can be avoided for example if the filter  $\mathbf{g}_\theta$  is represented as a fixed-order polynomial function of the matrix of eigenvalues  $\mathbf{\Lambda}$  [Defferrard et al., 2016].

#### Spatial-based ConvGNNs

The spatial-based methods are very similar to convolutional neural networks. One can think of them as generalizing the convolution operator from regular grids to graphs. They construct a new representation of a node by directly aggregating its representation and

the representations of the nodes in its neighbourhood. This approach directly acknowledges the spatial topology of the graph, and operates in the original domain of the node representations instead of the graph's spectral domain. The representations are usually transformed before the aggregation using a learnable function, in order to compute higher level features. The methods of this category are more popular than those of the spectral-based ConvGNNs. A method representative of the spatial-based ConvGNNs is presented more closely in Section 3.4.

## 3.2 Message Passing Neural Networks

After inspecting different graph neural network methods, the authors of [Gilmer et al., 2017] observed, that many specific implementations of graph neural networks can be encompassed using a single unified framework. This framework, called **Message Passing Neural Networks**, is closely related to the notions of spatial-based ConvGNNs. As its name suggests, it is based on a concept of messages, which the nodes pass between each other in order to update their representations.

The framework takes a graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  on the input, with node features  $\mathbf{x}_i$  and edge features  $\mathbf{e}_{ij}$ . It operates for  $T$  number of steps, during which the nodes' messages are generated and communicated through the graph. Each node gathers message sent to it by the nodes in its neighbourhood, and then utilizes the received information by accordingly updating its embedding.

The message sent to node  $i$  by the node  $j$  during step  $t$ , is generated using a learnable function  $M_t$  which takes as an input the current representations  $\mathbf{h}_i, \mathbf{h}_j$  of the two nodes and features  $\mathbf{e}_{ij}$  of the edge connecting them.

After the messages are generated and gathered, a parametrized function  $U_t$ , which is once again learned from the data, creates the node's new representation  $h_i^{t+1}$ .

$$\begin{aligned} \mathbf{m}_i^{t+1} &= \sum_{j \in \mathcal{N}(i)} M_t(\mathbf{h}_i^t, \mathbf{h}_j^t, \mathbf{e}_{ij}) \\ \mathbf{h}_i^{t+1} &= U_t(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}) \end{aligned} \quad (3.3)$$

Different specific graph neural network implementations can be then obtained by choosing different families of functions for the functions  $M_t$  and  $U_t$ .

After the message passing phase is completed, a representation of the entire graph can be constructed using the final hidden states of the nodes, if it is required for the task at hand. This representation is computed using a readout function  $R$ , which should be permutation invariant and also should accept inputs of an arbitrary size.

$$\hat{y} = R\left(\left\{\mathbf{h}_i^T \mid i \in \mathcal{G}\right\}\right) \quad (3.4)$$

The authors show that using this framework, it is easy to describe many different already existing GNN methods.

## 3.3 Graph Convolution Networks

One of the most popular convolutional graph neural network methods is the **Graph Convolution Network (GCN)** [Kipf and Welling, 2016]. The authors base their work in a spectral-based graph convolution network method [Defferrard et al., 2016], where the convolution filter  $\mathbf{g}_\theta$  is seen a function of eigenvalues  $\mathbf{\Lambda}$  of the graph's Laplacian, and is

approximated using a Chebyshev's polynomial of a fixed order  $K$ . The usage of Chebyshev's polynomial has been first suggested in [Hammond et al., 2011]. The authors of GCN propose to fix the order of the approximation to  $K = 1$ . This, along with other minor approximations and simplifications, yields a graph convolution operation that is expressed by the following equation

$$\mathbf{H}^{(l+1)} = \sigma \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right) \quad (3.5)$$

where  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ ,  $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ .

The authors originally intended this type of GNN to be used for transductive tasks (see 3.5), although it can generalize to induction tasks as well.

Note that by taking a first-order polynomial approximation of the graph filter  $\mathbf{g}_\theta$ , the method follows the principles of spatial-based ConvGNNs, even though it is based on a spectral-based approach. This is also apparent from the Equation 3.5. Therefore, one can see the spatial-based ConvGNNs as approximations of the spectral-based ConvGNNs.

### 3.4 Graph Attention Networks

Another quite popular method of the ConvGNNs category is the **Graph Attention Network (GAT)** [Veličković et al., 2017]. It introduces a graph attentional layer, that easily generalizes to neighbourhoods of various sizes while at the same time discerns the importance of individual neighbourhood nodes, thanks to the attention mechanism it uses to aggregate node features.

Let us inspect the inner workings of GAT. The GAT layer receives a set of node features  $\mathbf{h} = \{\mathbf{h}_1, \dots, \mathbf{h}_n\}$ ,  $\mathbf{h}_i \in \mathbb{R}^F$ , where  $n$  is the number of nodes and  $F$  is the dimensionality of the nodes' features. It outputs a transformed set of features  $\mathbf{h}' = \{\mathbf{h}'_1, \dots, \mathbf{h}'_n\}$ ,  $\mathbf{h}'_i \in \mathbb{R}^{F'}$ .

A linear transformation  $W \in \mathbb{R}^{F' \times F}$  is applied to each of the nodes' features, so that higher-level features can be obtained. A self-attention mechanism  $a$  is then applied to compute attention coefficients  $e_{ij}$  between pairs of nodes  $x_i$  and  $x_j$ . It is defined as

$$e_{ij} = \text{LeakyReLU} \left( \mathbf{a}^T [W\mathbf{h}_i \parallel W\mathbf{h}_j] \right) \quad (3.6)$$

where  $\parallel$  indicates the concatenation of two vectors and  $\mathbf{a} \in \mathbb{R}^{2F'}$  is a learnable projection vector.

The mechanism computes the attention coefficient  $e_{ij}$  for nodes  $i$  and  $j$  only if there exists an edge from  $i$  to  $j$ . The attention coefficients between nodes that are not directly computed are equal to zero. By doing so, GAT accounts for the graph's topology.

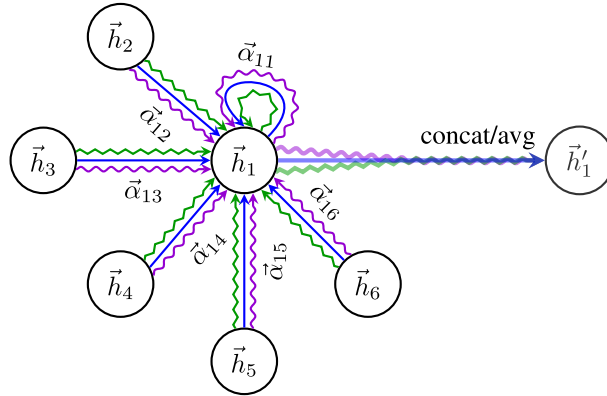
The attention coefficients are then normalized using the softmax function,

$$\alpha_{ij} = \text{softmax}_j (e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \quad (3.7)$$

where  $\mathcal{N}_i$  is the neighbourhood of the node  $i$ .

Finally, the resulting transformed node features  $\mathbf{h}'_i$  are the linear combination of the node's neighbourhood where the attention coefficients  $\alpha$  act as weights. A non-linearity  $\sigma$  is further applied to the linear combination.

$$\mathbf{h}'_i = \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij} W\mathbf{h}_j \right) \quad (3.8)$$



**Figure 3.1:** Figure showing the application of multi-head attention in GAT. The different styles of arrows signify computations of different heads. Reprinted from [Veličković et al., 2017].

The use of the attention mechanism allows GAT to assign different importances to neighbourhood nodes and therefore select which are more relevant to the query node, instead of weighing their contributions equally. It also allows it to naturally handle neighbourhoods of varying sizes, without receding to trivial approaches.

The authors also make use of the multi-head attention, similarly to [Vaswani et al., 2017]. The output of each individual head is computed as in equation 3.8. The authors then propose to either concatenate or average their outputs, according to what purpose they should serve further.

### 3.4.1 Modification of the Attention Mechanism

Despite GAT’s popularity and empirical evidence of its performance [Veličković et al., 2017], it has been found that the utilized attention mechanism has a hardly noticeable flaw, which nevertheless restricts its expressivity [Brody et al., 2021]. The computation presented in [Veličković et al., 2017] creates attention coefficients which are unconditioned on the query node.

This can be fixed by simply changing the order of operations in the computation of  $e_{ij}$  to

$$e_{ij} = \mathbf{a}^T \text{LeakyReLU}(W[\mathbf{h}_i \parallel \mathbf{h}_j]) \quad (3.9)$$

where now  $W \in \mathbb{R}^{F' \times 2F}$ .

The authors of this modification call the resulting layer **GATv2**. They have shown, both theoretically and experimentally, that **GATv2** is strictly more expressive than **GAT** and that the attention coefficients it computes are correctly conditioned on the query node.

## 3.5 Application of Graph Neural Networks

By using the graph neural networks, we derive new representations of the graph’s nodes. These are then utilized to solve different tasks. The tasks that GNNs solve can be classified according to the granularity level they are related to into **graph-level**, **node-level**, and **edge-level** tasks.

**Graph-level tasks.** These tasks are concerned about generating a prediction for the entire graph, be it a regression or a classification objective.

**Node-level tasks.** Node level tasks perform a prediction for each of the graph's nodes.

**Edge-level tasks.** Edge level tasks are related to individual edges of the graph. Apart from tasks concerning already existing edges, we can also predict for example the existence of an edge between each pair of nodes.

Furthermore, we can distinguish between **transductive** and **inductive** tasks, which, while being general terms in machine learning, are often a topic of discussion in the field of GNNs.

**Transductive tasks.** Transductive tasks require to perform the inference directly by reasoning from already observed data. An example of such a task would be to predict labels for unlabeled nodes based on graph nodes that have already been labeled.

**Inductive tasks.** Inductive tasks involve learning certain rules from a training dataset, and then being able to apply them to previously unseen data. An inductive task is for example the classification of a previously unseen graph.

For graph-level tasks, a final representation of the entire graph has to be constructed. One of the options to acquire this representation is to introduce an additional *global node*, that is connected to every node of the graph, and then use its final representation as the representation of the graph.

Other approach is to construct the graph's representation using the representations of individual nodes. To this effect, a pooling (or a readout) operation is applied. This pooling operation should be permutation-invariant, so that the order in which the nodes are presented does not affect the resulting representation. The simplest functions that have this property and a regularly used include the *sum*, *mean*, and *max* aggregation operators. These are usually applied to the final node embeddings, and do not take into account the topology of the graph.

More complex pooling methods have been introduced, for example **DiffPool** [Ying et al., 2018] or **SAGPool** [Lee et al., 2019]. They reduce the number of nodes in the graph and can be interleaved with GNN layers in order to construct hierarchical representations of the graph. Furthermore, these pooling methods can be trained jointly with the rest of the neural network.

After the final representations have been obtained, they are processed further to perform the specific task. For graph or node classification and regression tasks, a simple feed-forward neural network usually suffices.

### ■ 3.5.1 Application examples

Graph neural networks have been used for various tasks. The authors of [Stokes et al., 2020] used GNNs to predict the antibacterial activity of molecules. This allowed them to identify molecules that could have the potential to be used as new antibiotics. The



graph neural networks helped the authors discover one specific molecule, which they called Halicin. The effectivity of this molecule has then been shown also experimentally.

We can find examples of usage of GNNs also in the context of natural language processing. The authors of the Text Graph Convolutional Network (Text GCN) [Yao et al., 2019], use GNNs to classify documents contained within a corpus. They construct a single graph for the entire corpus, with nodes representing documents and single words. The text classification task is then expressed as a node classification task.



## Chapter 4

### Problem Statement

Transformers and their various modifications and derivatives quickly became the most prominent models in the field of NLP. Their ability to tackle different tasks while reusing the same general architecture and the ability to capture context and long range dependencies within text made them a rather popular choice for majority of the NLP tasks. Nevertheless, their usability is limited when it comes to processing long input sequences. As we already mentioned in Section 2.1.1, the computational complexity and memory requirements of the self-attention mechanism are  $\Theta(n^2)$ , where  $n$  is the number of input tokens. Therefore, one can see that due to this operation, the usage of Transformers becomes quickly unfeasible for longer sequences.

In order to alleviate this setback, different approaches were proposed as to how to deal with long input sequences. One of the easiest ones is to truncate the input sequence so that it does fit within the limits of the utilized model. This truncation can be executed differently based on the specifics of the task at hand. The most common way is to keep the first  $N_{\max}$  tokens, or concatenate different parts of the input sequence that are assumed to be important, like part of its beginning and part of its end.

Other techniques choose to modify the attention mechanism itself, and as a result decrease its computational complexity [Beltagy et al., 2020, Zaheer et al., 2020], while others choose to directly approximate the full attention mechanism [Choromanski et al., 2020, Xiong et al., 2021]. Such methods are then able to process much longer inputs.

Being able to process long input sequences and in doing so capture larger amount of information is quite relevant for some of the NLP tasks, such as question answering or summarization. Providing the model with a larger context has been shown to increase its performance on such tasks [Beltagy et al., 2020, Zaheer et al., 2020].

This thesis aims to make use of graph neural networks in combination with pre-trained language models to be able to handle the classification of input sequences of challenging lengths. Before we present our approach, we go over some of the existing methods.

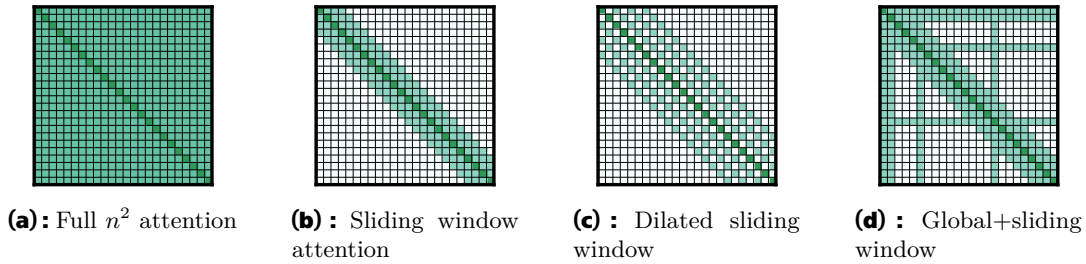
#### 4.1 Related Literature

We introduce some of the techniques aimed at processing long input sequences.

##### 4.1.1 Longformer

The Longformer [Beltagy et al., 2020] retains the architecture of the original Transformer. However, it modifies the inner workings of the utilized attention mechanism. Instead of allowing every token of the input sequence to attend to every other of the tokens, it only

allows the interactions of a limited number of certain tokens, chosen according to different attention patterns.



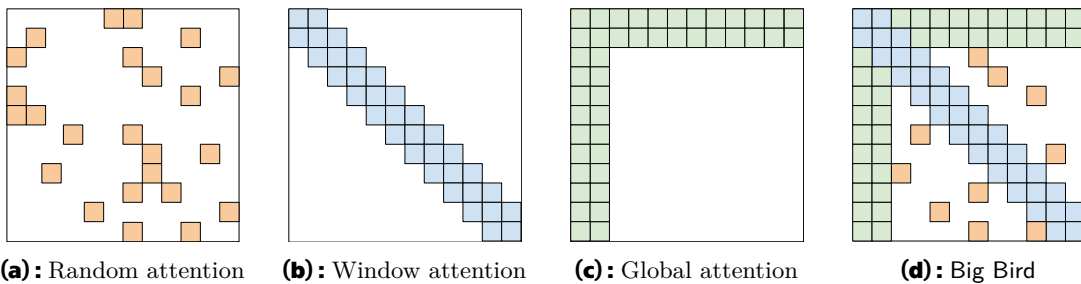
**Figure 4.1:** Comparison of the full attention mechanism pattern and the attention patterns introduced in the Longformer [Beltagy et al., 2020] paper. Reprinted from [Beltagy et al., 2020].

The attention patterns include for example a *sliding window* pattern, which allows each token to attend only to tokens in its neighbourhood of a constant size. The *dilated window* pattern is similar to the sliding window, but it introduces gaps between the attended tokens. Furthermore, the Longformer allows a predefined number of the tokens to attend to every other token. The different attention patterns are shown in Figure 4.1.

The restriction of the possible attention pairs effectively reduces the computational complexity and memory requirements of the attention mechanism from  $\Theta(n^2)$  to  $\Theta(n)$ . Therefore, it is easier for the Longformer to process long sequences, since the computational complexity and memory requirements increase only linearly with the length of the input sequence  $n$ . Even though tokens are not allowed to directly access all of the other tokens, stacking multiple sparse self-attention layers essentially increases the receptive window of the tokens.

This restriction of the attention mechanism can be also readily used in any of the already pre-trained Transformer-based language models.

#### 4.1.2 Big Bird



**Figure 4.2:** Comparison of the different attention patterns that make up the Big Bird’s sparse self-attention mechanism. Reprinted from [Zaheer et al., 2020].

Another model that is built around constructing a sparse attention mechanism is called Big Bird [Zaheer et al., 2020]. It is very similar to Longformer, as it also makes use of the sliding window pattern and specifies a predefined set of tokens with global attention. In addition to that, Big Bird also creates random connections between the tokens. This decision is motivated by graph theory, and its purpose is to decrease the average lengths of the shortest paths between pairs of tokens. This results in faster propagation of information through

different parts of the sequence. Furthermore, the authors prove that sparse attention mechanisms have theoretical properties very similar to that of the full quadratic attention mechanism and can be constructed so that they are universal approximators of sequence functions. Note that both **Big Bird** and **Longformer** can be likened to a special use case of graph neural networks and the attention patterns they introduce are essentially graph adjacency matrices.

### 4.1.3 Performer

The authors of Performers [Choromanski et al., 2020] note that many approaches aiming to reduce the computational complexity and memory requirements of the attention mechanism, such as **Longformer** and **Big Bird**, do not attempt to directly approximate the mechanism itself, but apply additional restrictions that result into different types of attention mechanisms that are not as resource demanding and easier to compute.

Therefore, they introduce the Performers architecture using a method that provably estimates the full self-attention mechanism without relying on any prior restrictions using a newly proposed mechanism called Fast Attention Via positive Orthogonal Random features (FAVOR+).

The authors approximate the (non-normalized) attention matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , which is originally computed using a dot product of the query and key vectors,

$$\mathbf{A} = \exp \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \quad (4.1)$$

as  $\mathbf{A}_{i,j} = K(\mathbf{q}_i, \mathbf{k}_j)$ , where  $K(\mathbf{x}, \mathbf{y}) = \mathbb{E} \left[ \phi(\mathbf{x})^T \phi(\mathbf{y}) \right]$  is a kernel function defined for some randomized mapping  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}_+^r$ .

By introducing transformed query and key matrices  $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{n \times r}$ , such that their rows are the original query and key vectors transformed by the randomized mapping  $\phi$ , the computation of the attention mechanism can be done efficiently as follows

$$\widehat{\text{Attention}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \widehat{\mathbf{D}}^{-1} \left( \mathbf{Q}' \left( (\mathbf{K}')^T \mathbf{V} \right) \right), \quad \widehat{\mathbf{D}} = \text{diag} \left( \mathbf{Q}' \left( (\mathbf{K}')^T \mathbf{1}_n \right) \right) \quad (4.2)$$

This approach allows to compute an approximation of the full self-attention mechanism in time and space complexity linear in terms of the sequence length, given that the computation order indicated by the parentheses in the formula is respected.

Furthermore, the authors propose that the random feature map  $\phi$  takes the following form

$$\phi(\mathbf{x}) = \frac{h(\mathbf{x})}{\sqrt{m}} \left( f_1(\omega_1^T \mathbf{x}), \dots, f_1(\omega_m^T \mathbf{x}), \dots, f_l(\omega_1^T \mathbf{x}), \dots, f_l(\omega_m^T \mathbf{x}) \right) \quad (4.3)$$

where  $f_1, \dots, f_l : \mathbb{R} \rightarrow \mathbb{R}$ ,  $h : \mathbb{R}^d \rightarrow \mathbb{R}$ , and  $\omega_1, \dots, \omega_m \in \mathbb{R}^d$  are chosen either deterministically or sampled i.i.d from some distribution  $\mathcal{D} \in \mathcal{P}(\mathbb{R}^d)$ . They also show the specific form of  $\phi$  for approximating the attention matrix. One of the important parts of FAVOR+ is to use  $\omega_i$  such that they are orthogonal, which strictly reduces the variance of the estimator.



## Chapter 5

### Proposed Solution

This work aims to utilize graph neural networks to be able to classify documents that exceed the size limitations of the Transformer-based models. As stated in Section 3, the GNNs are flexible and can represent various relationships. Therefore, as can be expected, there are various different options to make use of them to achieve our goal.

One option would be to treat each input token as a node and construct restricted connections between nodes. However, this would lead to models that are conceptually similar to those introduced in [Beltagy et al., 2020, Zaheer et al., 2020]. In fact, as mentioned in Section 3.2, GNNs represent a quite general framework and can be even used to model the original Transformer architecture. Therefore, we choose a different approach.

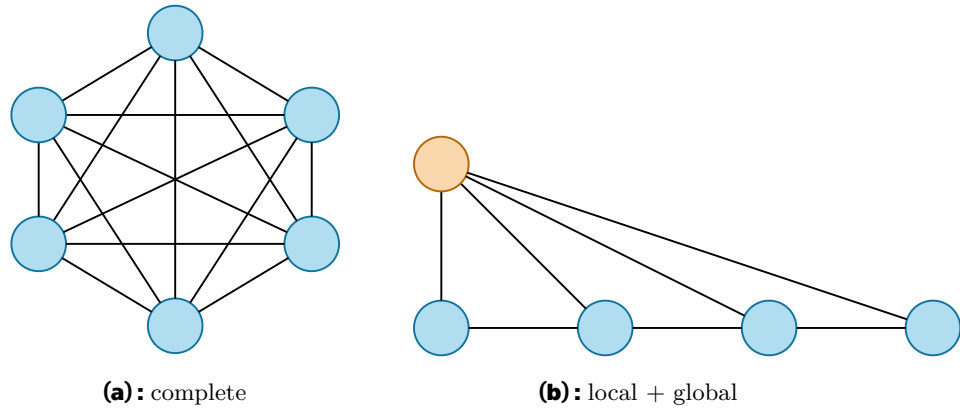
In this thesis, we utilize graph neural networks in combination with Transformer-based models. The text classification model that we present is based on processing parts of the input sequence separately. We split the input sequence into multiple parts, and initially process each of the parts individually. We create an embedding for every part using a pre-trained BERT-like model, which can be obtained for example from the Hugging Face model repository. The parts of the input are then thought of as nodes of a graph, and their embeddings are used as the features of these nodes. Nodes of textual graphs were previously encoded using embedding vectors for example in [Hamilton et al., 2017]. Having created this graph, we use GNNs to allow for interactions between different encoded parts of the input sequence. Finally, the text classification is seen as a graph classification task. An encoding of the entire graph is obtained through a pooling operation and then utilized to perform the final classification. The entire process is illustrated in Figure 5.2.

This approach remains quite general and can be applied to different specific text classification tasks. The way the input is separated into individual parts can be tailored to the task at hand.

This method makes use of the valuable knowledge that is already contained within the pre-trained language models, such as BERT and its derivatives [Devlin et al., 2018, Liu et al., 2019]. Once trained, it is possible to scale this model quite simply to inputs of various sizes. Note that this method can be also quite easily modified to perform classifications at the level of individual parts of the input, by classifying the corresponding nodes of the graph.

One of the important aspects of this approach is the decision as to which nodes should be connected with an edge. We propose two different ways of creating the edges between the parts of the input. The first one of these is to create a connection between each pair of nodes, thus making a complete graph. This allows the nodes to directly exchange messages, without any necessary intermediary. On the other hand, it disregards the ordering of the input parts and therefore is permutation invariant. An illustration of this topology is shown in part a) in Figure 5.1.

The second approach involves creating connections between nodes that correspond to



**Figure 5.1:** Figure showing the two proposed graph topologies. Section a) of the figure shows a *complete* graph with six nodes, and section b) shows the *local + global* topology configuration with the additional latent node, which is shown in orange. Self-loops are not shown for clarity.

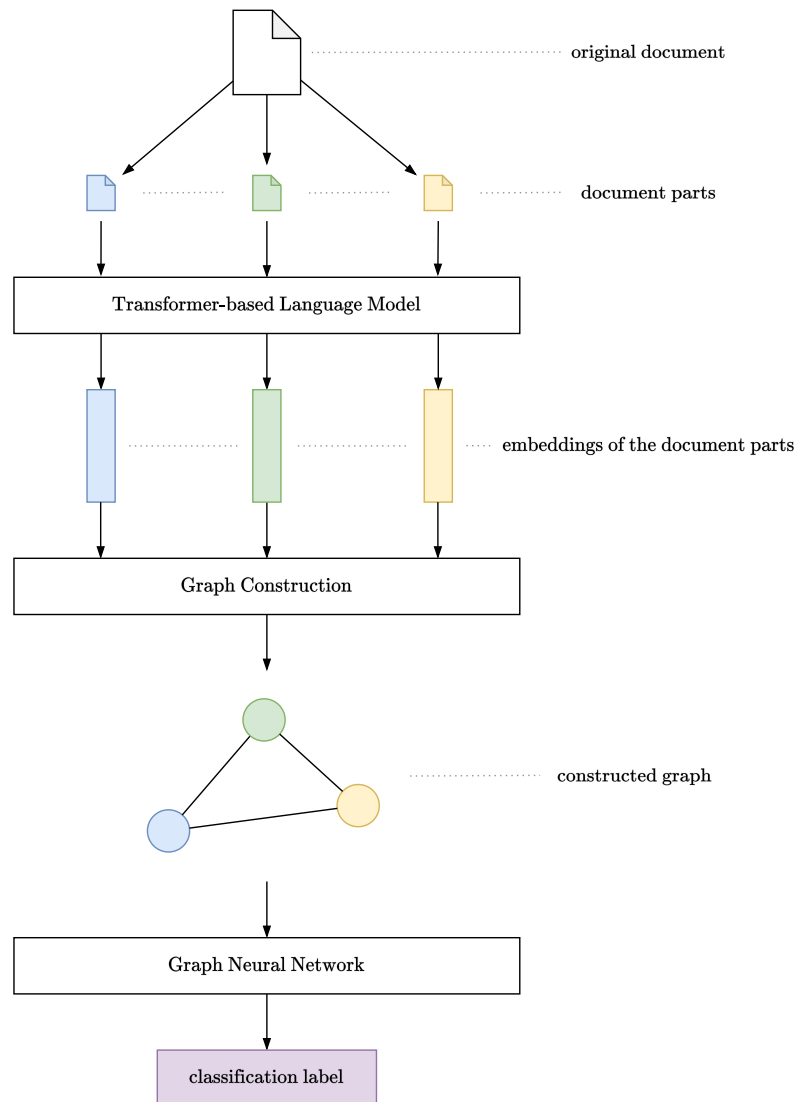
parts of the input that appear next to each other in the input sequence. Furthermore, an additional latent node is created and connected to every other node of the graph. Nearby parts of the input are therefore allowed to communicate directly with each other, and all nodes can exchange messages indirectly through this additional node. The embedding of the latent node is initialized as a mean of the embeddings of the original graph nodes. We call this topology configuration *local + global*. As opposed to the *complete* graph configuration, it is not permutation invariant, and the order of the inputs affects which input parts are connected with an edge. A small example of a *local + global* graph with 4 nodes is shown in part b) in Figure 5.1.

Both of these topology configurations are quite general and straightforward. Therefore, no assumptions are being made about the nature of the specific underlying classification task.

The implementation of the proposed model is publicly available.<sup>1</sup>

<sup>1</sup><https://github.com/aic-factcheck/long-input-gnn>





**Figure 5.2:** An overview diagram of the proposed model. The input document is first split into multiple parts. An embedding is created for every part using a Transformer-based language model. These embeddings are then used as features of the nodes in the constructed graph, which connects different parts of the input. Finally, a graph neural network composed of multiple GNN layers is used to obtain the final classification of the input document.



# Chapter 6

## Datasets

### 6.1 MultiSource

Since we found it difficult to obtain a text classification dataset that would contain a considerable amount of examples (satisfying the data-intensive needs of the Transformer-based architectures) and at the same time would be composed of examples with input sequences of challenging lengths, we decided to propose our own text classification task and also to create a custom dataset for this task.

The goal of the proposed task, which we call **MultiSource**, is to detect whether the text of the input sequence originates from a single source document, or whether it is composed of parts that come from different documents.

The nature of this task allows us to generate custom datasets. To do that, it is required that we have access to a collection of documents, using which the individual examples will be constructed. For our purposes, we chose to use the English Wikipedia snapshot, originally meant for the Fact Extraction and Verification (FEVER) task [Thorne et al., 2018]. It includes only the abstracts of the Wikipedia entries.

The by-product of being able to generate the datasets is the ability to control the length of the dataset’s examples.

The process of generating the datasets for the **MultiSource** task is quite general and highly customizable. New dataset instances based on various corpora can be made quite easily. The task itself might be considered in some sense similar to that of the *Next Sentence Prediction*, introduced in Section 2.2.

#### 6.1.1 Dataset Generation Process

The process of generating a **MultiSource** dataset goes as follows. We treat the documents from the utilized collection at the level of sentences. The *train*, *validation*, and *test* splits of the dataset are generated individually. Each split is composed of multiple parts. For each of these parts, it is possible to define the number of sentences  $S_n$  that each of their examples should contain, and the number of sentences  $S_{\text{random}} \leq S_n$  that should originate from different sources. Furthermore, every part defines the number of examples  $N$  it should contain.

Then, to generate a single example of a part, we sample an initial original document  $d_{\text{orig}}$  from an available set of documents  $\mathcal{D}$ , such that it has at least  $S_n - S_{\text{random}}$  sentences. From this document,  $S_n - S_{\text{random}}$  sentences are sampled. These sentences form the base of the example. Subsequently,  $S_{\text{random}}$  other documents are sampled from  $\mathcal{D}$ , and a single sentence is randomly selected from each of them. These sentences are added to those already selected from the original document. The resulting set of sentences forms the

complete generated example. The label of the example is determined by the number of random sentences  $S_{\text{random}}$ , with examples where  $S_{\text{random}} > 0$  being labeled as positive.

Finally, the sentences are randomly permuted. This is done to encourage the classifier to truly learn to compare different parts of the text and not simply detect possible abrupt changes in the structure of the text.

The sentences from the original article are sampled *without replacement*. This prevents the occurrence of some unintentional cues that the classifier might be able to detect, such as the same sentence occurring multiple times within a single example.

---

**Algorithm 1** Generating a part of a MultiSource dataset split.

---

```

1: procedure GENERATEMULTISOURCESPLIT( $\mathcal{D}$ ,  $N$ ,  $S_n$ ,  $S_{\text{random}}$ )
2:   examples  $\leftarrow \emptyset$ 
3:   for  $i = 1$  to  $N$  do
4:     example  $\leftarrow \emptyset$ 
5:      $d_{\text{orig}} \leftarrow \text{sample\_document}(\mathcal{D}, \text{min\_sentences} = S_n - S_{\text{random}})$ 
6:     example  $\leftarrow \text{example} \cup \text{sample\_sentences}(d_{\text{orig}}, \text{n\_sentences} = S_n - S_{\text{random}})$ 
7:     for  $j = 0$  to  $S_{\text{random}}$  do
8:        $d_{\text{random}} \leftarrow \text{sample\_document}(\mathcal{D}, \text{min\_sentences} = 1)$ 
9:       example  $\leftarrow \text{example} \cup \text{sample\_sentences}(d_{\text{random}}, \text{n\_sentences} = 1)$ 
10:    end for
11:    examples  $\leftarrow \text{examples} \cup \{\text{example}\}$ 
12:  end for
13: end procedure

```

---

The process of generating a part of the dataset’s split is also described by pseudo algorithm 1.

To prevent data leakage between the individual dataset splits, we track which documents were used for generating individual splits. This information is then used when generating a new dataset split, with documents used to generate the previous splits being removed from the set of available documents  $\mathcal{D}$ .

The complete dataset for the MultiSource task is then the composition of the *train*, *validation*, and *test* splits.

### 6.1.2 Document Sampling

The way we select the documents from which the random sentences are sampled greatly influences the resulting difficulty of a particular MultiSource dataset. During our initial inspections of the generated datasets, we quickly noticed that sampling the documents uniformly at random generates examples that are quite easy to classify.

Therefore, we decided to add an option to sample documents that are more similar to the original document  $d_{\text{orig}}$ . This modification makes the positive generated examples harder to detect and makes the task more challenging. We say that such examples were generated using a *similar* sampling.

To retrieve similar documents, we have utilized the Anserini toolkit for information retrieval [Yang et al., 2017, Lin et al., 2021]. For every original document  $d_{\text{orig}}$ , we would run a query to retrieve the top  $r$  documents from the *complete* collection of documents. These documents would then be filtered so that only those present in the currently available set of documents  $\mathcal{D}$  are kept. Out of the remaining documents, we sample uniformly from the top  $k$  most similar to obtain the random documents. It is possible that there are

no similar documents such that they were not yet used in other dataset splits. In that case, a random document is sampled from the available set without paying respect to its similarity. Nevertheless, such cases were almost non-existent and occurred only rarely.

The process of generating a MultiSource dataset allows us to modify its specific characteristics and in effect gives us some control over the difficulty of the task. One of the options is to use the sampling of similar documents, as we have just described. The other is being able to directly specify the number of random sentences that should appear within positive examples.

The configuration of the positive examples can, for example, be pushed to the extreme, where only a single sentence would originate from a different source document, which would additionally be sampled out of documents similar to the original one. In such instances, the task could even be thought of as a within-text anomaly detection.

```

1 {
2   "label": true,
3   "ids": ["Robert_Punkenhofer", "Robert_Punkenhofer", "
↪ Julius_von_Bismarck", "Salvador_Mas_Conde"],
4   "sentences": [
5     "He currently serves as the Austrian Trade Commissioner in
↪ Barcelona, a job he has also performed in Mexico City, Berlin and
↪ New York City.",
6     "Robert Punkenhofer (born July 7, 1965 in Austria) His career
↪ intertwines art, design and architecture as well as international
↪ business development.",
7     "This equipment is positioned in a public place to capture and
↪ subsequently airbrush pigeons as they chart their path within the
↪ vicinity.",
8     "Born in Barcelona, the conductor Salvador Mas-Conde started his
↪ musical studies in Escolaria de Montserrat, continuing at the
↪ Conservatorio Superior Municipal de Música of Barcelona (CSMM), in
↪ Salzburg with Bruno Maderna, in Siena with Franco Ferrara, and at
↪ the Vienna University of Music and Dramatic Art with Hans
↪ Swarowsky and Günther Theuring."
9   ]
10 }
```

**Figure 6.1:** A small MultiSource example generated using the similar sampling. It is composed of four sentences, where two come from a single identical source, and the other two sentences originate from two different sources.

An example generated using a similar sampling is shown in Figure 6.1, and Figure 6.2 shows an example generated using a purely random document sampling.

```

1 {
2   "label": true,
3   "ids": ["Agnes_Campbell", "Bold_hypothesis", "Agnes_Campbell", "
↪ Mohamed_Ayoub_Ferjani"],
4   "sentences": [
5     "At that time O'Neill had been supporting the English.",
6     "The concept is nowadays widely used in the philosophy of
↪ science and in the philosophy of knowledge.",
7     "She helped mobilise Scottish support for the Irish.",
8     "He will compete in Rio de Janeiro with his brother Fares, who
↪ qualified in men's sabre."
9   ]
10 }

```

**Figure 6.2:** A small MultiSource example. The sentences originating from different sources were sampled from documents obtained using random sampling, without regards to the ‘original’ source document.

We use the described generation process to create several different MultiSource datasets, which are further described in Section 7.2.

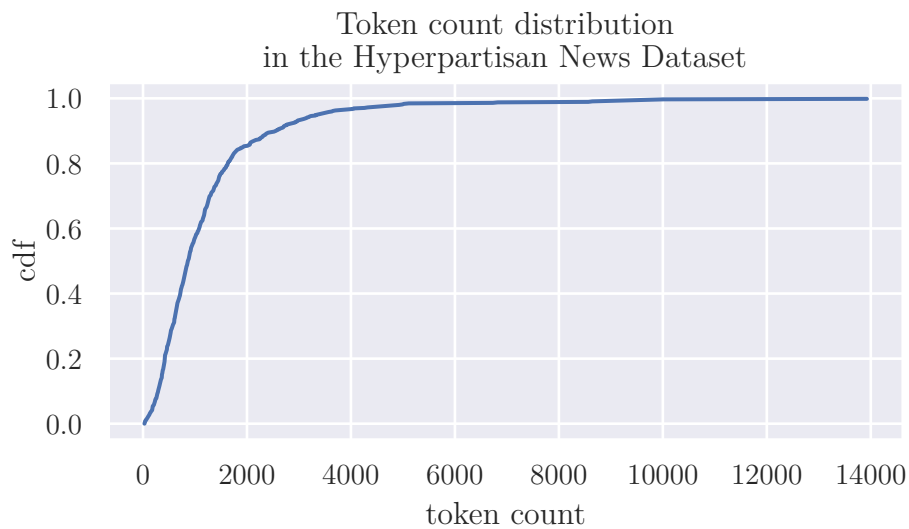
## 6.2 Hyperpartisan News Detection

The Hyperpartisan News Detection dataset [Kiesel et al., 2019] is composed of articles from various online news websites. The goal is to detect articles whose content shows strong support for a single political party and disregards any opposing opinions or views. The information provided in these articles can be presented in a way that is quite misleading. The form of the used language is often such as to provoke a strong emotional response in the reader.

The reason we chose this dataset, apart from being an interesting task, is that the token count of the majority of the examples significantly exceeds the most common limitations of models with Transformer-based architecture. The median of the examples’ token counts is 863, and the longest input sequence has 13,926 tokens. Therefore, we will be able to test the ability of our model to handle truly long sequences. The cumulative distribution function of the token counts is shown in Figure 6.3. However, the size of the dataset is quite small, having only 645 examples. Furthermore, it is necessary to be cautious when using this dataset to train classification models since its class distribution is not balanced. We have split the dataset into training, validation, and test splits. The splits have been obtained using stratified sampling, retaining the label distribution of the complete dataset. Their individual label distributions are shown in Table 6.1. The news articles are provided in an HTML format. A single example from the dataset is shown in Figure 6.4.

dataset split	label	
	positive	negative
train	164	281
validation	37	63
test	37	63

**Table 6.1:** The class distributions for different splits of the Hyperpartisan News Detection dataset.



**Figure 6.3:** The cumulative distribution function of the token counts of the Hyperpartisan News Detection dataset examples. Most of the examples contain more than 512 tokens. The median length of an example is 863 tokens, and the dataset contains examples as long as 13,926 tokens.

```

1 {
2   "title": "WATCH: This Man Tells The Truth! Exposes politicians
   ↪ creating opportunities for violence",
3   "label": 1,
4   "url": "https://politicalcowboy.com/watch-man-tells-truth-exposes-
   ↪ politicians-creating-opportunities-violence/",
5   "published_at": "2017-08-24",
6   "text": "<p>Yet again we see more evidence that the media and
   ↪ leftist government officials are lying to the American public.
   ↪ What is the agenda? What reality are they try to protect? Why are
   ↪ they risking violence when it is unnecessary?</p> \n<p>Watch this
   ↪ video posted on Facebook by Will Johnson, a common sense
   ↪ conservative, who intends to be a part of a free speech rally this
   ↪ weekend in San Francisco. As he states it is intended to be a pro
   ↪ Trump rally, however it is clearly being painted by politicians
   ↪ and the media as a white supremacist rally with a call for the
   ↪ presence of Antifa. How is it possible to have a white supremacy
   ↪ rally when the rally barely has any white people attending?</p> \n
   ↪ <p>I'll let you figure that one out! Watch the video below.</p> \n
   ↪ <p>https://www.facebook.com/newlypress/videos/1875854002678972/</p>
   ↪ > 60 Celebrities Who Are Mega Trump Supporters - #20 Is
   ↪ Unbelievable!"
7 }

```

**Figure 6.4:** An example of a hyperpartisan news article from the Hyperpartisan News Detection dataset.

## 6.3 CTKFacts

Last but not least, we introduce the CTKFacts dataset [Drchal et al., 2022], a Czech fact-checking dataset. The fact-checking task takes a written statement, a *claim*, as its input and utilizes a knowledge base in form of a simple database of text documents according to which it attempts to verify the veracity of the statement. It is one of the tasks where the usage of natural language processing could be beneficial and help with its automation. It is then not that surprising that systems for automated fact checking have been gaining a considerable amount of attention [Thorne and Vlachos, 2018].

The dataset was collected by the FactCheck group at AIC CTU in collaboration with the Department of Journalism at the Faculty of Social Sciences of Charles University. The collection process and further cleaning and quality inspections resulted in a dataset composed of 3,097 original manually annotated textual claims in Czech. An example of a claim with accompanying evidence is shown in Figure 6.5. We refer the reader to the original research paper [Drchal et al., 2022] for specific information about the process of the data collection.

---

**Claim:** Spojené státy americké hraničí s Mexikem.

**EN:** The United States of America share borders with Mexico.

---

**Evidence 1:** “Mexiko a USA sdílejí 3000 kilometrů dlouhou hranici, kterou ročně překročí tisíce Mexičanů v naději na lepší životní podmínky (...)”

**EN:** *Mexico and the U.S. share a 3,000-kilometer border, crossed by thousands of Mexicans each year in hopes of better living conditions (...)*

**Evidence 2:** “Mexiko také nelibě nese, že Spojené státy stále budují na vzájemné, několik tisíc kilometrů dlouhé hranici zeď, která má zabránit fyzickému ilegálnímu přechodu Mexičanů do USA (...)”

**EN:** *Mexico is also uncomfortable with the fact that the United States is still building a wall on their mutual, several thousand kilometers long border, to prevent Mexicans from physically crossing illegally into the U.S. (...)*

---

**Figure 6.5:** An example of a claim from the training split of the CTKFacts dataset, along with its evidence. The claim is supported by the information contained in the knowledge base. The example is reprinted from [Drchal et al., 2022].

The dataset itself is inspired by the FEVER dataset [Thorne et al., 2018], and we utilize a similar approach to claim verification as the one used by the authors of FEVER. The verification is done using an automated fact checking pipeline, which is composed of two main steps; document retrieval and natural language inference (NLI). Since the textual claims are not paired with their respective documents that might be used to verify them, a set of evidence documents has to be retrieved from the knowledge base. This is the purpose of the document retrieval step. Once the evidence is retrieved, it is passed along with the claim to a classification model, which performs the NLI step. The output of this automated fact checking system is then a label classifying the textual claim into one of three classes, specifying its veracity according to the information available in the evidence retrieved from the knowledge base. The specific labels are SUPPORTS, REFUTES, and NOT ENOUGH INFORMATION (NEI). The distribution of labels in individual splits of the dataset is shown in Table 6.2. This thesis focuses on the NLI part of the fact checking pipeline. The documents to be used as evidence have been already retrieved and sorted by their relevance

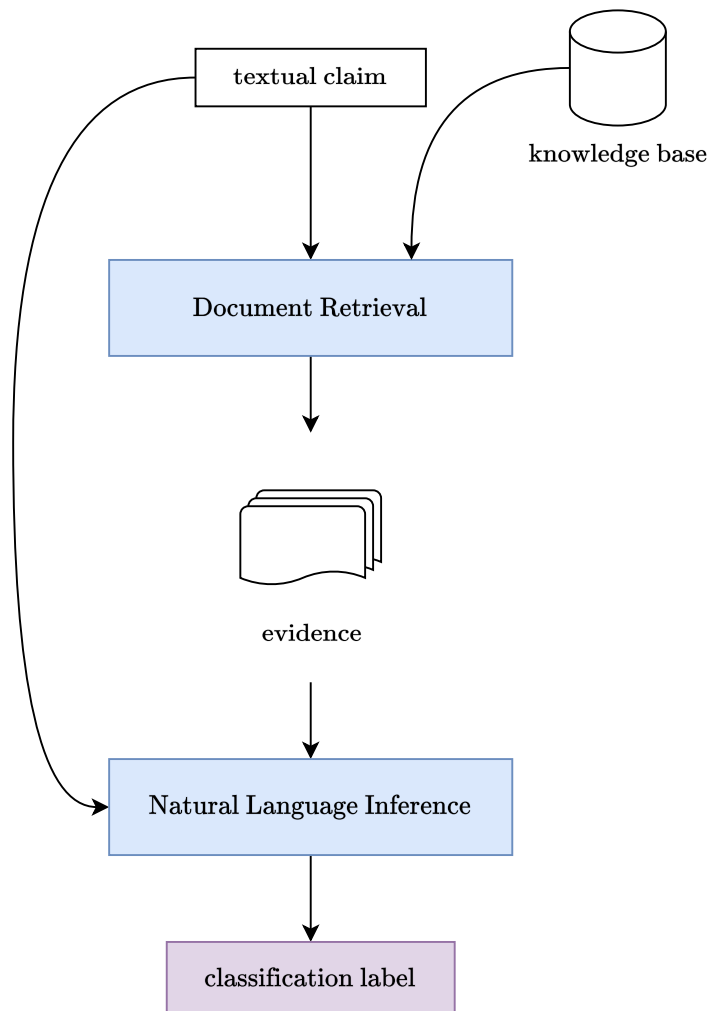


using the Anserini toolkit [Yang et al., 2017]. A diagram of the automated fact checking pipeline is shown in Figure 6.6.

dataset split	label		
	SUPPORTS	REFUTES	NEI
train	1,104	556	723
validation	142	85	105
test	176	79	127

**Table 6.2:** The class distributions for different splits of the CTKFacts dataset.

Even though the dataset takes inspiration from the FEVER dataset, it uses a different type of knowledge base, which it then regards as the ground truth when verifying the claims. The FEVER dataset makes use of Wikipedia entries, whereas CTKFacts uses an archive of news articles published by the Czech News Agency (Česká Tisková Kancelář).



**Figure 6.6:** The automated fact checking pipeline. The model proposed in this thesis performs the natural language inference step. The classification label can be one of SUPPORTS, REFUTES, and NOT ENOUGH INFORMATION.



# Chapter 7

## Experiments

We have carried out number of experiments using the datasets presented in Chapter 6. The first experiments are done using the **MultiSource** task, where we examine the properties of our model as well as of the task itself. After that, we continue with the **Hyperpartisan News Detection** dataset, which provides a challenge in form of long inputs and a limited number of examples. Finally, we test the performance of our model on the **CTK Facts** dataset.

### 7.1 Experimental Setup

Our proposed model has a number of configurable parameters, specified in the following list, that need to be chosen accordingly.

- **Type of the utilized language model.** Our model allows to use different language models to create the embeddings of individual parts of the input. This choice can play an important role, based on the language of processed data and even on the nature of the task.
- **Type of the GNN layers.** Different types of GNN layers can be used. Nevertheless, we restrict our experiments to the usage of the GATv2 [Brody et al., 2021] layer, which will allow the model to decide the importance of different input parts during GNN message propagation.
- **Number of GNN layers.** The number of GNN layers will control the size of the neighbourhood in which different nodes can affect the representations of other nodes. In our experiments, we consider using 1 to 4 layers.
- **Number of heads per GNN layer.** Since the GAT layer allows us to use multi-head attention, we can also control the number of heads being used. The outputs of individual heads are concatenated and then transformed using a linear projection, similarly to the approach chosen in the original Transformer architecture. We experiment with 1, 2, and 4 heads.
- **Size of the node embeddings.** We can control the size of the embeddings of the graph nodes. After computing the embeddings of the input parts using the language model, the GNN layers project them into a linear space of this size. This size is then same for every GNN layer. We refer to this parameter also as the hidden size.
- **Topology of the graph.** The topology of the graph will specify which of the nodes, and consequently parts of the input, can exchange information directly.

- **Pooling operation.** The pooling operation can be any function, which is permutation invariant. We restrict our experiments to usage of the simple *max* pooling operation.

Throughout our experiments we use the following pre-trained language models or their previously fine-tuned versions. We utilize the *base* architectures of these models.

- **BERT.** The plain BERT model described in Section 2.2.
- **mBERT.** The multilingual version of BERT, trained on the WIKIPEDIA dumps of the top 100 languages with the most WIKIPEDIA pages. This exposes the model to different languages, including Czech, during its pre-training.
- **RoBERTa.** A model with an architecture identical to that of BERT, however, it is pre-trained on a much larger dataset with larger batches and its pre-training does not include the *Next Sentence Prediction* task [Liu et al., 2019]. Furthermore, the masking for the MLM task is done dynamically during the training, as opposed to BERT’s pre-training, which generates the masks beforehand.
- **RoBERTa @ SQuAD2.** A RoBERTa model that has been additionally fine-tuned using the SQuAD2 task [Rajpurkar et al., 2018].
- **XLM-RoBERTa.** A multilingual version of the RoBERTa model, pre-trained using a dataset containing texts in 100 different languages. It was introduced by [Conneau et al., 2019].
- **XLM-RoBERTa @ SQuAD2.** The XLM-RoBERTa model fine-tuned using the SQuAD2 task.
- **FERNET-C5.** A BERT model targeted for Czech language, pre-trained on a corpus of Czech entries extracted from the Common Crawl project. It was introduced by [Lehečka and Švec, 2021].
- **RobeCzech.** A RoBERTa model pre-trained exclusively on Czech data, introduced by [Straka et al., 2021].

We train the models using the AdamW optimizer [Loshchilov and Hutter, 2017] with learning rate  $\alpha = 2 \times 10^{-5}$  and weight decay  $\lambda = 0.01$ . During the training of a model, we track its performance on the validation split of a dataset, and store the model parameters that resulted in the best value of the performance criterion. As not all of the datasets used within this thesis are balanced, we select and report different performance metrics accordingly.

Due to the large number of possible configurations of our model, we either restrict our choices or select certain hyperparameters independently in a greedy manner. The specific approach is described for each of the experiments.

### ■ 7.1.1 Baselines

In order to be able to evaluate the performance of our model, it is required that we have baseline results that we can compare to. The baselines chosen here are Transformer-like models without any modifications. The input is truncated in case its size exceeds the limitations of the specific model. The model architecture of the Transformer-like baselines corresponds to the architecture used for the encoding of the parts of the input in our own model, i.e., if BERT is used for obtaining the embeddings of different parts of the input, then the baseline to which comparisons are made is a fine-tuned BERT model.

### 7.1.2 Implementation Details

The utilized Transformer models and input tokenizers have been obtained through the Hugging Face model repository [Wolf et al., 2020]. Our model has been implemented in PyTorch with the help of the PyTorch Geometric library [Fey and Lenssen, 2019], which provides tools for deep learning on graphs and contains implementations of many graph neural network methods. We also used the PyTorch Lightning library for the ease of development and training of the models. The implementation of the model is publicly available<sup>1</sup>.

## 7.2 MultiSource

We use differently configured MultiSource datasets in order to try and ascertain the behaviour of our model and explore the difficulty of the MultiSource task. During these experiments, we mostly restrict our model to the use of the base BERT model for computing the input embeddings.

The generated datasets were created so that the size of individual examples is within the limitations of the used pre-trained language models, and therefore the baseline models are able to utilize the entire input sequences. Each of the datasets contains only examples with 10 sentences, however the specific configuration of the examples can differ.

When using our model, we treat the examples at the level of individual sentences and each sentence represents a single node of the generated graph. This will allow us to see how GNNs are able to combine information from different parts of the input.

### 7.2.1 Hyperparameters selection

In order to select how many GNN layers should our model have and how many multi-attention heads should be used by GATv2, we chose perform a grid search over multiple values of these two hyperparameters. During these experiments, the graph topology was fixed to the *complete* graph setting, and the hidden size was set to equal 256. We expect that the performance of our model will not match that of the baseline, as the baseline Transformer model will be able to process the entire input directly, whereas our approach will have to process each sentence individually before it can relate them all to each other.

We generated a dataset which we call MultiSource-Mix-40k. It contains a mixture of examples with different numbers of random sentences, obtained using the similar sampling approach. All of the examples have a total of 10 sentences, and the number of positive and negative examples is balanced within each of the dataset splits. The specific configuration of this dataset is shown in Table 7.1.

---

<sup>1</sup><https://github.com/aic-factcheck/long-input-gnn>

random sentences count	split		
	train	validation	test
0	20,000	1,000	1,000
2	5,000	250	250
4	5,000	250	250
6	5,000	250	250
8	5,000	250	250

**Table 7.1:** Configuration of the MultiSource-Mix-40k dataset splits. The individual entries indicate the number of examples.

The resulting test accuracy of the different model configurations trained on the MultiSource-Mix-40k dataset is shown in Table 7.2.<sup>2</sup>

layers count	heads count		
	1	2	4
1	77.85	79.75	79.45
2	76.45	79.70	80.35
3	77.45	78.60	78.05
4	75.80	<b>80.80</b>	79.65

**Table 7.2:** The test set accuracies of our model with different choices for the graph neural network hyperparameters on the MultiSource-Mix-40k dataset. The baseline test accuracy, obtained using the BERT language model, was 89.55%.

The results of this grid search show that there is a larger increase in performance when increasing the number of multi-attention heads from 1 to 2 for all of the layer counts. However, increasing the number of heads further does not provide any additional advantage, and in most cases even hinders the classifier’s performance. There does not seem to be any noticeable trend in the performance with respect to the number of GNN layers. Nevertheless, the best performing classifier had 4 GATv2 layers with 2 attention heads. We chose to use this configuration in the subsequent experiments for both the *complete* and *local + global* topologies.

It is important the note, that even the best performing configuration achieved a significantly lower test accuracy in comparison to the baseline obtained using the BERT model, which was 89.55%.

Furthermore, we also tested the effect of changing the size of the node embeddings. The test accuracy results shown in Table 7.3 indicate that using smaller hidden sizes negatively affects the performance. Increasing it to 512 gives a similar performance as when we set it to 256, and therefore the results do not show much potential for further improvement with increasing the hidden size. As a result, we chose to keep the size of the node embeddings at 256.

<sup>2</sup>Correctly, the comparison of the configurations would have been done using the validation split performance instead of the test one. Nevertheless, this experiment serves only to select a base configuration for further experiments, where we utilize newly generated datasets in majority of the cases.

node embedding size	64	128	256	512
test accuracy	76.85	76.65	<b>80.80</b>	79.85

**Table 7.3:** The influence of the node embedding’s size on the classifier’s performance. Results shown are the test accuracies.

## 7.2.2 Task Difficulty Analysis

Since we propose the MultiSource task as a part of this work, we wanted to ascertain its complexity and at the same time observe the behaviour of our model under different circumstances. To this effect, we generated different MultiSource datasets with specific configurations, each designed to observe a different aspect of the task and its requirements.

The first property of the MultiSource task that we inspected was the influence of the random sentences sampling strategies. As mentioned in 6.1.2, the usage of the similar sampling strategy should result in examples that are hard to classify, due to the similarity between the sampled sentences. On the other hand, sampling the sentences purely at random with no regards to their similarity should give examples where the differences between different sources should be much easier to detect.

For this purpose, we generated a dataset with a configuration identical to that of MultiSource-Mix-40k, with the only difference being that we utilized the purely random sampling strategy.

sampling strategy	model	
	our model	BERT
purely random	96.55	99.30
similar	80.80	89.55

**Table 7.4:** Results of an experiment showing the influence of the sampling strategy on the difficulty of the MultiSource task. Along with the test accuracy of our model, we also show the test accuracy of the BERT baseline.

The outcome of the experiment is shown in Table 7.4. It is clear that the sampling strategy greatly impacts the difficulty of detecting examples composed of parts originating from different sources. When the purely random sampling is used the baseline model has little problem with learning to classify the examples, classifying correctly 99.7% of them. When we change the sampling strategy to the similar one, we can immediately see a drop in the performance of the baseline model. The same pattern is repeated for our proposed model as well, where it achieves an accuracy of 96.55% for the purely random sampling strategy and an accuracy of 80.8% for the similar sampling one.

random sentences count	1 sentence		2 sentences	
	complete	local + global	complete	local + global
2	<b>63.90</b>	63.50	<b>72.85</b>	67.30
4	<b>75.70</b>	71.20	83.15	<b>83.40</b>
6	<b>81.25</b>	80.65	<b>90.70</b>	87.30
8	<b>87.40</b>	86.45	<b>92.30</b>	91.75

**Table 7.5:** Results of the experiment inspecting the influence of the ratio of sentences from a single source and sentences from different sources. Shows comparison between different topologies and the effect of using a sliding window over sentences.

Another thing that can be controlled during the generation of the examples is the number of random sentences from different sources contained within the examples. It is quite logical to expect that when the ratio of sentences from different sources to sentences from the original source is large, the difficulty of the task is easier than when it is quite small. Looking for a single out of place sentence in a set of ten similar sentences can be likened to looking for a needle in a haystack.

Due to the nature of this task, one can expect that the performance of the classifier will improve if the encoder is given direct access to larger parts of the input. We check if this property can be observed by creating parts of the input using a sliding window over the sentences of the input. We set the size of the window to 2 sentences and keep the stride equal to a single sentence. Each node of the graph will therefore represent a pair of consecutive sentences.

We inspect the influence of the ratio of random sentences and of the encoding of pair of sentences jointly in a single experiment. We create multiple MultiSource datasets, each with a different configuration. Each of them has a different predefined number of random sentences contained within its positive examples. That way, we can observe how the classifier behaves according to different ratios of sentences from a single source and from multiple sources. The results of this experiment are shown in Table 7.5.

The experiment confirms our expectations, and the performance of the classifier increases with the number of sentences from different sources. Furthermore, we can see that most of the time the *complete* topology achieves better results than the *local + global* topology. According to our expectations, using a 2-sentence sliding window helps the classifier achieve higher accuracy.

### 7.2.3 Importance of Graph Neural Networks

When using the graph neural networks as we do in our model, it is quite natural to ask to what extent do they influence the performance of the classifier. Therefore, we conducted an ablation study to test exactly that. In order to do that, we tested how our model will perform in case we simply do not create any edges between the nodes of the graph, and keep only the nodes' self-loops. That will prohibit the model from exchanging information between different parts of the input. Each part will therefore be processed individually by the Transformer-based encoder as well as by the GNN model built on top of it. Note that further transformations using feed-forward neural network layers, identical in structure (but not parametrization) to those that are applied when the edges are created, *will* be applied to the embeddings, and only the ability to propagate the information is restricted.

Furthermore, we try whether further increasing the size of the moving window has any effect, and how will its usage affect the performance of the model when no edges are



window size	topology			
	complete	local + global	local	none
1 sentence	<b>80.80</b>	76.45	74.35	56.40
2 sentences	<b>84.80</b>	83.85	83.10	81.20
3 sentences	84.85	<b>85.85</b>	85.40	84.75
4 sentences	87.60	87.40	<b>88.05</b>	87.15

**Table 7.6:** The results of the graph neural networks ablation experiment. The best results for each of the window sizes are shown in bold.

created.

We also try removing the latent *global* node in the *local + global* topology. This modified topology is called simply *local*, and it connects only the nodes that represent neighbouring parts of the input.

Table 7.6 shows the results of this experiment on the MultiSource-Mix-40k dataset. We can see that when the parts of the input and consequently the node embeddings are formed by single sentences, the model has the best performance with the *complete* topology. The second best performance is achieved by the *local + global* topology. Note that its accuracy decreases when the *global* node is removed, as seen in the result of the *local* topology. When no edges between the nodes are created, the classifier does not perform much better than if it was guessing at random. Therefore, we can see that in this case, the graph neural networks truly allow the model to combine information coming from different parts of the input sequence.

Nevertheless, it is apparent that when the size of the input used for generating node embeddings is increased, the differences between the topologies diminish. This can be somewhat explained by the nature of the task. It is reasonable to expect that once the Transformer-based encoder has access to multiple sentences, it is able to detect whether these sentences originate from different sources. If it then includes this information in the embeddings of the nodes, it can be easily collected by the final graph pooling operation. This would effectively decrease the importance of the chosen topology, and in fact even the importance of using graph neural networks, and would be in line with what we have seen in this experiment.

## 7.3 Hyperpartisan News Detection

We inspect the performance of our classifier on the Hyperpartisan News Detection dataset. Since this dataset has imbalanced positive and negative classes, we assign different weights to each of them when computing the loss during the training of a model. The weights are calculated so that the sums of weights per class are equal. We use the F1 score as the performance metric.

We choose a simple approach to dividing the text into multiple parts. We separate it into parts that have the largest length such that they can still be processed by the language model utilized for creating their embeddings.

Once again, we begin by inspecting the hyperparameters of our model. Given the size of the dataset, we were able to perform the grid search for both of the *complete* and *local + global* topologies separately. We use BERT as the model’s encoder to create the node embeddings, and the hidden size is set to 256. The results of these grid searches are shown in Tables 7.7 and 7.8 for the *complete* and *local + global* topologies, respectively.

layers count	heads count		
	1	2	4
1	80.00	80.60	77.50
2	76.54	<b>84.93</b>	75.68
3	82.19	76.19	76.71
4	76.47	77.29	80.56

**Table 7.7:** The F1 validation scores for different choices of the GNN settings for the *complete* topology.

The best configuration found for the *complete* topology was 2 GATv2 layers with 2 heads, achieving an F1 score of 84.93 on the validation set. On the other hand, the *local + global* topology performed the best with 3 layers and only a single head. It achieved an F1 score of 83.78.

Furthermore, we also try changing the size of the node embeddings, however do not find any improvement over the already selected size.

layers count	heads count		
	1	2	4
1	82.19	81.69	80.00
2	81.69	78.26	80.00
3	<b>83.78</b>	75.68	77.11
4	73.17	68.29	78.95

**Table 7.8:** The F1 validation scores for different choices of the GNN settings for the *local + global* topology.

### 7.3.1 Language Models Comparison

We also inspect the influence of the used language model. We compare the achieved performance when the BERT language model is swapped for RoBERTa when creating the embeddings of an article’s parts. The number of GNN layers and number of heads are selected according to the previous experiment. We consider both the *complete* and *local + global* topologies. The results of this comparison are shown in Table 7.9. Despite our expectations, using RoBERTa to obtain encodings of parts of the input did not improve the performance of the classifier. Therefore, we continue using BERT as the encoder of the parts of the input.

topology	encoder	
	BERT	RoBERTa
<i>complete</i>	<b>84.93</b>	82.93
<i>local + global</i>	<b>83.78</b>	82.67

**Table 7.9:** Comparison of F1 scores on the validation split when the BERT model is swapped for RoBERTa. Interestingly enough, using RoBERTa does not yield better results.

### 7.3.2 Baseline Comparison

Finally, we compare the resulting test F1 scores achieved by our models and the baseline BERT classifier in Table 7.10. Since the F1 score does not provide us with a complete picture of the classifiers’ performance, we also report the precision and the recall of our best performing model configurations and the baseline model. We can see that our model with the *local + global* topology outperforms the BERT classifier. On the other hand, the *complete* topology did not yield very good results, achieving an F1 score of only 73.42. Compared to the other classifiers, it lacks the ability to correctly identify the positive examples, as it has the lowest precision score.

We also wanted to compare the performance of our model to that of the Longformer. Unfortunately, the composition of the dataset splits used by Longformer’s authors is not made exactly clear and we had to create our own dataset splits. Therefore, we fine-tuned a Longformer classifier ourselves using its pre-trained checkpoint from the Hugging Face repository. Similarly to the authors of this language model, we applied global attention to the [CLS] token. The fine-tuned Longformer outperformed every other model.<sup>3</sup> Nevertheless, our proposed model with the *local + global* topology was able to almost match its performance in terms of the F1 metric.

model	metric		
	F1	precision	recall
BERT	77.14	81.82	72.97
Longformer	<b>80.60</b>	<b>90.00</b>	72.97
<i>complete</i>	73.42	69.05	78.38
<i>local + global</i>	80.00	78.95	<b>81.08</b>

**Table 7.10:** The resulting F1 test scores. Our proposed model with the *local + global* topology outperforms the baseline BERT model and matches the Longformer in the F1 metric.

Note that these experiments also show that our model is able to process inputs as long as 13,296 tokens, which is the longest example contained within this dataset.

## 7.4 CTKFacts

The fact checking dataset introduces a new challenge by containing only Czech input, which is a low resource language. This means it would be difficult to use for example the original BERT model for encoding the documents, because it has been pre-trained on an English corpora, and might not generalize that well to Czech. Therefore, we make use of language models which have been pre-trained on data that contains Czech text. These include for example mBERT and a fine-tuned version of XLM-RoBERTa. Furthermore, we also test the FERNET-C5 and RobeCzech models, both of which have been pre-trained exclusively on Czech data.

We also have to decide how exactly to process the claims along with their retrieved evidence sets using our model. An initial idea one might come up with is to treat the claim and every evidence as a single part of the input. However, this might make it difficult to convey the relationship between the claim and the evidence, or even to learn

<sup>3</sup>The authors of Longformer achieved significantly better results on *different* splits of this dataset, observing an F1 validation score of 94.8.

that the “claim” node should be interpreted differently with respect to the “evidence” nodes as opposed to how the “evidence” nodes treat each other.

Therefore, we decided to make use of BERT’s ability to process a pair of input sequences, separated using the special [SEP] token. Each one of the retrieved evidence  $e_i$  is encoded together with the factual claim  $c$  as a pair of sequences  $(c, e_i)$ . This allows the encoder to directly relate the inspected claim with the evidence. These encodings are then used to create the graph. This approach can also be said to be motivated by the idea that at first, the language model extracts information relevant to the claim from the evidence, which is then further combined by the graph neural network.

The input is processed similarly for the baseline methods, only the evidence set is concatenated and possibly trimmed by the tokenizer.

The usage of graph neural networks has a consequence which might have an impact on the performance of our classifier. Recall that the automated fact checking pipeline operates in two consecutive steps, the document retrieval and natural language inference (see Figure 6.6). The document retrieval module returns an evidence set for a given claim. Along with this set, it also provides an ordering of the evidence, communicating which of the evidence it considers as being more important for the actual fact checking. This information is preserved when the classification is done using the Transformer-based models, since we can concatenate the evidence according to this ordering. However, in Chapter 3 we stated that GNNs for graph-level tasks are permutation invariant. This fact, combined with the usage of *complete* topology, results in the loss of the evidence importance information. This information is also *partially* lost when using the *local + global* topology, with the evidence of consecutive importance being represented by neighbouring nodes in the graph.

It is also important to note that we perform our experiments in the *full pipeline setting*, which means that we utilized evidence sets retrieved by the method for document retrieval, and not sets of golden evidence as indicated by the dataset’s annotators. As a result, the quality of the predictions is inherently limited by the quality of the retrieved evidence.

We would also like to note that we noticed that our resulting approach is similar to that presented in [Zhou et al., 2019].

### 7.4.1 Initial Experiments

Our experiments for the CTKFacts carry on in a similar fashion to those conducted as part of the previous experiments. During these experiments, we make use of the top 10 evidence from the retrieved evidence set.

layers count	heads count			layers count	heads count		
	1	2	4		1	2	4
1	58.73	56.93	59.94	1	<b>57.23</b>	50.60	50.30
2	57.83	58.73	<b>60.24</b>	2	55.42	51.20	53.31
3	<b>60.24</b>	57.23	58.73	3	51.51	46.99	47.59
4	57.83	54.22	54.82	4	49.70	51.51	49.10

(a): mBERT

(b): XLM-RoBERTa @ SQuAD2

**Table 7.11:** Comparison of validation F1 micro scores when using the cross-lingual language models, mBERT and XLM-RoBERTa @ SQuAD2, as encoders along with different configurations of the GNNs with *complete* topology.

layers count	heads count			layers count	heads count		
	1	2	4		1	2	4
1	61.75	60.84	63.86	1	61.75	55.72	59.34
2	62.35	<b>64.16</b>	57.23	2	60.84	58.73	56.02
3	62.35	61.14	62.65	3	<b>64.46</b>	59.64	56.63
4	63.55	58.43	58.13	4	61.75	42.77	54.52

**(a):** FERNET-C5 **(b):** RobeCzech

**Table 7.12:** Comparison of validation F1 micro scores when using the Czech monolingual language models, FERNET-C5 and RobeCzech, as encoders along with different configurations of the GNNs with *complete* topology.

We begin by inspecting the classifier’s performance on validation set for different configurations of the graph neural network in *complete* topology setting for all of the language models we set out to test. The results for the cross-lingual models and the Czech monolingual models are summarized in Tables 7.11 and 7.12, respectively. They indicate that out of the considered models, the ones trained exclusively on Czech data are better accustomed for this task. Interestingly enough, XLM-RoBERTa @ SQuAD2 seems to perform the worst, even though it has shown the best performance during the initial NLI experiments in [Drchal et al., 2022].

Based on these results, we consider only FERNET-C5 and RobeCzech for the encoder of our model in further experiments. We also examine the performance using the *local + global* topology, and find that it yields slightly better but comparable results for FERNET-C5, although with different configuration of GNNs, as shown in Table 7.13. On the other hand, it hinders the classifier’s ability when RobeCzech is used. This result seems somewhat curious, as one might expect a similar change in performance for both of the encoders when the graph’s topology is changed.

layers count	heads count			layers count	heads count		
	1	2	4		1	2	4
1	59.94	59.94	60.84	1	58.73	59.64	59.04
2	58.13	61.45	<b>65.06</b>	2	<b>62.05</b>	59.34	56.93
3	61.45	63.25	59.94	3	57.83	56.02	54.22
4	59.94	58.73	62.65	4	59.34	53.31	54.22

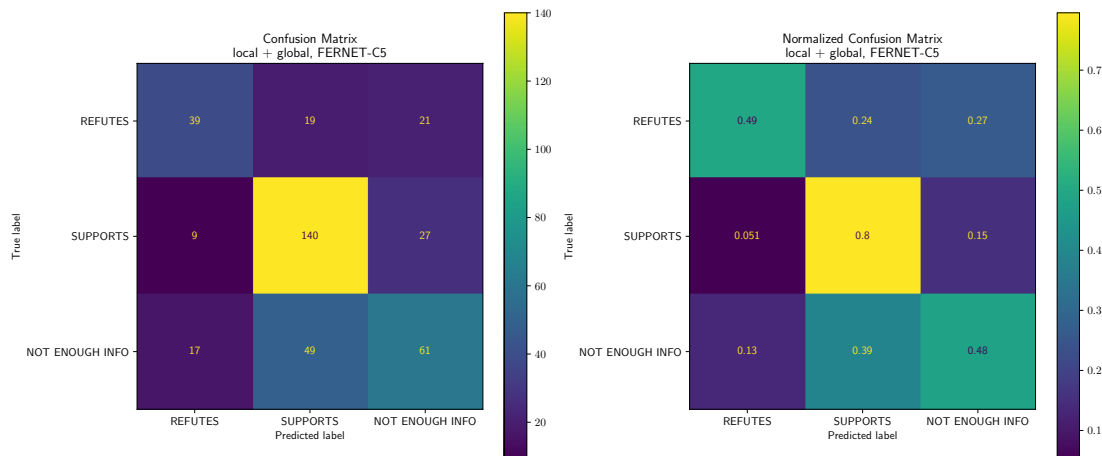
**(a):** FERNET-C5 **(b):** RobeCzech

**Table 7.13:** Comparison of CTKFacts validation F1 micro scores for different configurations of GNNs with *local + global* topology and different language models.

Finally, we compare the best performing model configuration with the baseline FERNET-C5 model in Table 7.14. It appears that our classifier outperformed our chosen baseline in both F1 micro and macro metrics. Nevertheless, it achieved only comparable results to that of the best classifier presented by the authors of CTKFacts, which achieved an F1 macro score of 59.69 [Drchal et al., 2022].

model	metric	
	F1 micro	F1 macro
FERNET-C5	58.12	54.32
<i>local + global</i>	<b>62.83</b>	<b>59.59</b>

**Table 7.14:** Comparison of our best performing model configuration in the *complete* and *local + global* topology settings and the FERNET-C5 baseline.



**Figure 7.1:** Confusion matrix for the test set of CTKFacts. The predictions were obtained using our proposed model with FERNET-C5 as the encoder, *local + global* topology, and 2 GATv2 layers with 4 heads. The node embedding size was set to 256.

Figure 7.1 shows the confusion matrix and its normalized version for the predictions of our classifiers. It seems that it can identify claims that are supported by the evidence the best, whereas it struggles when it comes to the other two classes.

We also find it important to note that after performing an ablation experiment similar to that carried out in Section 7.2.3, we observed that the introduction of a topology connecting different parts of the input provided only a *small benefit* over a model which did not create any edges between the nodes. The model without node connections achieved a test F1 micro and macro scores of 62.04 and 58.44, respectively, which is a performance comparable to that of the classifier utilizing a *local + global* topology.

The benefits of using graph neural networks might prove to be greater in a multi-hop setting, where it is necessary to combine information from several sources in order to make a correct prediction. Unfortunately, we were not able to examine this behaviour using CTKFacts (or any other of our datasets), since it contains only a small number of multi-hop examples.

## 7.4.2 Changing the Amount of Available Evidence

During our initial experiments, we utilized the top 10 evidence retrieved from the knowledge base. We inspect the influence of the amount of provided evidence on the performance of our classifier. Increasing the amount might possibly introduce additional unnecessary noise, since the retrieved evidence does not have to be closely related to the claim that we are fact checking. On the other hand, restricting the amount of information available

to the model might possibly hinder its performance as well. Therefore, we compare the performance of our classifier when 5, 10, 15, and 20 most semantically relevant evidence as provided by the document retrieval method are used.

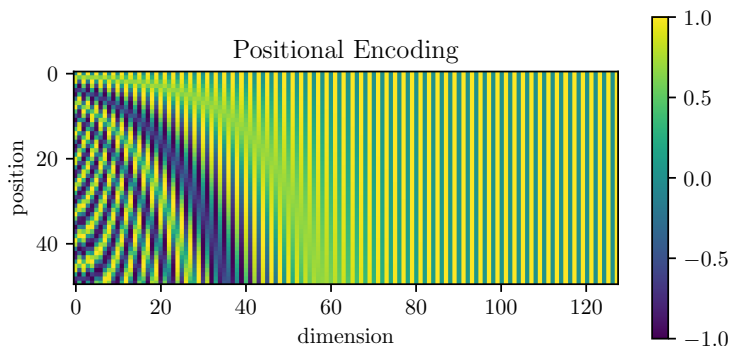
evidence count	metric	
	F1 micro	F1 macro
5	61.26	57.62
10	<b>62.83</b>	<b>59.59</b>
15	60.21	56.06
20	60.21	55.45

**Table 7.15:** The resulting CTKFacts test set performances of our classifier for different numbers of provided evidence.

As can be seen in Table 7.15, neither decreasing nor increasing the amount of evidence available improved the performance of the classifier. We would like to note that the median token counts of the claims paired with their concatenated evidence sets for 10, 15, and 20 evidence were 708, 1,056, and 1,416, respectively.

### 7.4.3 Embedding the Importance of the Evidence

As mentioned in the introduction to Section 7.4, the information about evidence importance provided by the document retrieval method is either completely disregarded or at least partially lost when using our model in combination with the proposed topologies. We attempt to alleviate this shortcoming by embedding the importance as a positional information into the embeddings of the different input parts generated by the language model. We adopt a technique for positional encoding identical to that introduced by the authors of the Transformer architecture [Vaswani et al., 2017]. An example of the positional encodings is shown in Figure 7.2. We generate the positional encoding  $\mathbf{p}_i$  for every evidence  $e_i$  corresponding to its position in the retrieved set and modify its embedding  $\mathbf{x}_i$  generated by the Transformer-based model by adding the two, obtaining its representation with embedded position  $\mathbf{x}'_i = \mathbf{x}_i + \mathbf{p}_i$ .



**Figure 7.2:** An example of 128-dimensional positional encodings, generated for the first 50 positions.

positional encoding	metric	
	F1 micro	F1 macro
no	<b>62.83</b>	<b>59.59</b>
yes	59.42	57.11

**Table 7.16:** The influence of embedding the evidence importance using positional encoding.

We test this modification with our best performing configuration of the classifier, and present the results in Table 7.16. Unfortunately, including the information about the evidence importance did not help us achieve better results. It is possible that embedding the importance using this approach is not correct, and only introduces additional noise in the encoding of the input.



## Chapter 8

### Discussion

During our experiments, we have not observed our model achieving any significant improvements over the selected baseline methods, and at times it was not even able to match their performance. Because of that, we discuss what might be the reasons behind this fact, identify some of the drawbacks of our proposed method, and propose further research possibilities.


One of the possible limiting factors that we identified is the embedding of parts of the input into a single vector. This can be likened to the case of recurrent neural networks, where the usage of a single vector to represent an input sequence was one of the reasons that hindered their performance. The disadvantage of relying on this single vector has been shown through the use of the attention mechanism [Bahdanau et al., 2014]. Nevertheless, it was also the main property of our model which allowed us to process long input sequences.

Another weak point of our proposed method is that it treats the input parts independently during the generation of their embeddings. This can result in the loss of certain information, where parts of the input might not be considered important for the embedding until they are combined with a different part of the sequence. Therefore, this information could be lost during the encoding stage. It is possible that this could be partially mitigated through an approach similar to that used in [Yang et al., 2021], where graph neural network layers are interleaved with self-attention layers, thus entangling the encodings of different sequences.

Furthermore, we made limited use of the flexibility of graph data during our experiments by restricting ourselves to only two simple topologies, the *complete* topology and the *local + global* topology. However, one might propose more involved methods for generating the graph connecting different input parts. For example, edges between different parts might be created based on their semantic similarities. We have assumed that the attention mechanism of GATv2 will infer the importance of the connections. Another possibility might be to treat the input on different levels of granularity and create its hierarchical representation, similarly to [Fang et al., 2019]. For example, one could create separate nodes for the sentences of the input, and nodes for different paragraphs of the input. The “sentence” nodes could then be connected to their respective “paragraph” node, and “paragraph” nodes could be connected using a complete graph.

Nevertheless, we were able to successfully reuse the *already pre-trained* language models for processing long inputs. This made it possible to process long sequences, for example in Czech, without having to pre-train a model such as Longformer from scratch using Czech data. However, the importance of graph neural networks should be investigated more thoroughly based on the results from Sections 7.2.3 and 7.4.1. It is possible that they act only as an aggregator of partial results, as proposed in Section 7.2.3.





## Chapter 9

### Conclusion

This thesis set out to construct a classifier of sequences that exceed the limitations of the Transformer-based architecture, while at the same time making use of the already pre-trained language models. We presented the field of natural language processing, including the Transformers architecture and its original motivation. We gave an overview of deep learning on graphs using graph neural networks, presented their different categories, and introduced a few specific implementations. We also studied and described some of the already existing approaches for processing long input sequences.

We proposed and implemented a model that is able to process long input sequences by dividing the input sequence into multiple parts, embedding each individually using a pre-trained language model and further processing these embeddings using a graph neural network. We proposed two general approaches for creating the graph's topology.

We introduced three different tasks, one of which, the **MultiSource** task, we presented as a part of this work. Datasets for this task can be constructed from an arbitrary corpus, and their difficulty can be controlled by configuring the process of their generation. We used the different tasks to assess the performance of our classifier, compared it to the baseline models, and performed various experiments. We have shown that our model is able to process inputs as long as 13,296 tokens.

Finally, we discussed the properties of our model, identified some of its weak points, and proposed possible future work.





## Bibliography

- [Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- [Beltagy et al., 2020] Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.
- [Brody et al., 2021] Brody, S., Alon, U., and Yahav, E. (2021). How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.
- [Brown et al., 1990] Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85.
- [Carion et al., 2020] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer.
- [Chen et al., 2020] Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. (2020). Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR.
- [Chen et al., 2021] Chen, X., Wu, Y., Wang, Z., Liu, S., and Li, J. (2021). Developing real-time streaming transformer transducer for speech recognition on large-scale dataset. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5904–5908. IEEE.
- [Choromanski et al., 2020] Choromanski, K., Likhoshesterov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. (2020). Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.
- [Conneau et al., 2019] Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*.
- [Defferrard et al., 2016] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29.
- [Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

- [Drchal et al., 2022] Drchal, J., Ullrich, H., Rýpar, M., Vincourová, H., and Moravec, V. (2022). Csfever and ctkfacts: Czech datasets for fact verification. *arXiv preprint arXiv:2201.11115*.
- [Fang et al., 2019] Fang, Y., Sun, S., Gan, Z., Pillai, R., Wang, S., and Liu, J. (2019). Hierarchical graph network for multi-hop question answering. *arXiv preprint arXiv:1911.03631*.
- [Fey and Lenssen, 2019] Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.
- [Gilmer et al., 2017] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.
- [Gulati et al., 2020] Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., et al. (2020). Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*.
- [Hamilton et al., 2017] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.
- [Hammond et al., 2011] Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150.
- [Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- [Kiesel et al., 2019] Kiesel, J., Mestre, M., Shukla, R., Vincent, E., Adineh, P., Corney, D., Stein, B., and Potthast, M. (2019). Semeval-2019 task 4: Hyperpartisan news detection. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 829–839.
- [Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.
- [Lee et al., 2019] Lee, J., Lee, I., and Kang, J. (2019). Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. PMLR.
- [Lehečka and Švec, 2021] Lehečka, J. and Švec, J. (2021). Comparison of czech transformers on text classification tasks. In *International Conference on Statistical Language and Speech Processing*, pages 27–37. Springer.
- [Lin et al., 2021] Lin, J., Ma, X., Lin, S.-C., Yang, J.-H., Pradeep, R., and Nogueira, R. (2021). Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2356–2362.
- [Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

- [Liu et al., 2021] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022.
- [Loshchilov and Hutter, 2017] Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.
- [Radford et al., 2022] Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision. Technical report, Tech. Rep., Technical report, OpenAI.
- [Rajpurkar et al., 2018] Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don’t know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.
- [Sandryhaila and Moura, 2013] Sandryhaila, A. and Moura, J. M. (2013). Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656.
- [Shuman et al., 2013] Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98.
- [Stokes et al., 2020] Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z., et al. (2020). A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.
- [Straka et al., 2021] Straka, M., Náplava, J., Straková, J., and Samuel, D. (2021). Robeczech: Czech roberta, a monolingual contextualized language representation model. In *International Conference on Text, Speech, and Dialogue*, pages 197–209. Springer.
- [Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.
- [Thorne and Vlachos, 2018] Thorne, J. and Vlachos, A. (2018). Automated fact checking: Task formulations, methods and future directions. *arXiv preprint arXiv:1806.07687*.
- [Thorne et al., 2018] Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. (2018). Fever: A large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*.
- [Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- [Veličković et al., 2017] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *stat*, 1050:20.
- [Wolf et al., 2020] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

- [Wu et al., 2020] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.
- [Xiong et al., 2021] Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. (2021). Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14138–14148.
- [Yang et al., 2021] Yang, J., Liu, Z., Xiao, S., Li, C., Lian, D., Agrawal, S., Singh, A., Sun, G., and Xie, X. (2021). Graphformers: Gnn-nested transformers for representation learning on textual graph. *Advances in Neural Information Processing Systems*, 34:28798–28810.
- [Yang et al., 2017] Yang, P., Fang, H., and Lin, J. (2017). Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 1253–1256.
- [Yao et al., 2019] Yao, L., Mao, C., and Luo, Y. (2019). Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377.
- [Ying et al., 2018] Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31.
- [Zaheer et al., 2020] Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. (2020). Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297.
- [Zhou et al., 2019] Zhou, J., Han, X., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2019). Gear: Graph-based evidence aggregating and reasoning for fact verification. *arXiv preprint arXiv:1908.01843*.