Westfälische Hochschule

University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Masterarbeit

Titel der Arbeit // Title of Thesis

Vergleich von Dependenz- und Konstituenzgrammatik bei syntaxangereicherten Transformern
Comparison of dependency and constituency grammar with syntax-enhanced transformers

---

Akademischer Abschlussgrad: Grad, Fachrichtung (Abkürzung) // Degree
Master of Science (M.Sc.)

Autorenname, Geburtsort // Name, Place of Birth
Caren Daniel, Köln

Studiengang // Course of Study
Wirtschaft und Informationstechnik

Fachbereich // Department
Informatik: Intelligente Systeme

Erstprüferin/Erstprüfer // First Examiner
Prof. Dr. Gregor Kroesen

Zweitprüferin/Zweitprüfer // Second Examiner
Prof. Dr. Peter Nalbach

Abgabedatum // Date of Submission
31.10.2022

Westfälische Hochschule
University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Eidesstattliche Versicherung

Daniel Caren
Name, Vorname // Name, First Name

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem Titel

Vergleich von Dependenz- und Konstituenzgrammatik bei syntaxangereicherten Transformern

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Brühl, den 27. Oktober 2022
Ort, Datum, Unterschrift // Place, Date, Signature

Abstract

Transformer models such as BERT may profit from already available data such as linguistic syntax graphs to improve their predictions. In this thesis, a syntax-enhanced BERT model (SynGNN) is built, using two popular types of syntax graphs: dependency grammar and constituency grammar-style syntax trees. The syntax graphs are processed using a Graph Neural Network (GNN). Previous work such as Sachan et al. (2021) on which the architecture used here is based, only investigate the performance of dependency grammar.

The conducted experiments suggest that with a task of Named Entity Recognition (NER) on a small dataset the SynGNN model outperforms the pre-trained BERT with both types of syntax trees. Also, it performs better when using constituency-style syntax graphs than with dependency-style graphs.

Normally, such linguistic syntax graphs are hand-annotated and costly to produce. So, another point of interest was the question whether it is possible to replace the handmade graphs with automatically generated ones without sacrificing model performance. It was found that this is indeed a viable approach, when using high-quality automatic parsers the performance is almost the same. This means that it may be worthwhile to use the information provided by these syntax graphs in an automated model architecture and for any kind of text.

Contents

List of Figures

List of Tables

1 Introduction

BERT is a machine learning system designed for a variety of Natural Language Processing (NLP) tasks. Studies such as Luo (2021) or Goldberg (2019) have shown that the BERT attention heads are able to learn syntactic information similar to that which is encoded in linguistic syntax graphs describing the syntactic structure of a sentence. However, depending on the dataset BERT may not always be able to extract all relevant data.

As such, these syntax graphs can be a useful resource for the model to learn additional relationships. They are available for several, hand-annotated datasets and come in two main types. The first are dependency grammar-style syntax graphs and the second constituency grammar style syntax graphs. These two styles put their focus on different aspects, differ in their representation and also in the exact information encoded in them.

To efficiently harvest the information stored in the syntax graphs, Graph Neural Networks have recently emerged as a means to process graph structures. So, it seems natural to try and combine the capabilities of BERT with additional prior data in the form of syntax graphs filtered through a GNN.

So far, efforts to enhance BERT with syntactic information mostly used dependency style-syntax graphs and the representation called constituency grammar is mostly ignored. Also, there exist no concise comparisons of both syntax styles when used for NLP tasks. This work aims to create a *SynGNN*: a BERT-based model enhanced with syntactic information filtered through a GNN. The model performance is then evaluated with hand-annotated dependency-style and constituency-style syntax graphs.

The accompanying code for this thesis is available at: `https://github.com/shrdlu-whs/syngnn`

Constituent	Head	Dominates
Noun Phrase (*NP*)	*NN* - cat	*DT* - the, *JJ* - large
Verb Phrase (*VP*)	*VBZ* - is	*VP*
Verb Phrase (*VP*)	*VBG* - chasing	*NP*
Noun Phrase (*NP*)	*NN* - ball	*DT* - a

Table 1: Sentence constituents

2 Linguistic Groundwork

In this chapter the linguistic core concepts used for the model are introduced with a short overview of constituency and dependency grammar and their respective advantages and drawbacks for use in Natural Language Processing (NLP) systems.

2.1 Constituency Grammar

Constituency grammar is one of the most common formal models of syntactic structure in linguistics. Its core idea is that sentences are not only sequences of separate words. Instead, they are made up of *constituents* as building blocks, which behave as if they were single units. Each constituent, also called a *phrase* gets its name from its *head* which dominates all other elements in the phrase. For an overview, refer to Fromkin et al. (2013). Consider the constituency tree in figure 1 for the sentence *“the large cat is chasing a ball”*. The table in 1 lists all constituents as well as their respective heads and dominance relations for the sentence. For example, a Noun Phrase such as *the large cat* has the noun *cat* as its head, which dominates the determiner phrase DT, *the* and the adjective phrase JJ, *large*.

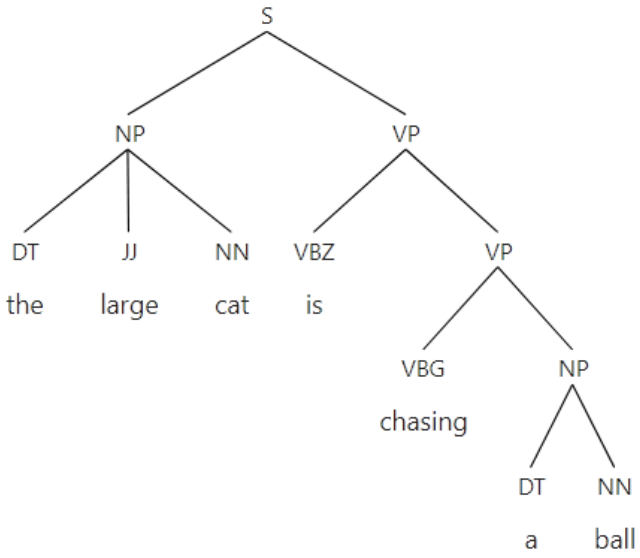


Figure 1: Constituency tree for 'the large cat is chasing a ball' - generated by Benepar implementation from Stern (2022)

Constituency grammar was first formalized by Chomsky (1956/1975) using the concept of context-free grammar (CFG). A CFG consists of a 4-tuple (N, Σ, R, S) with N the set of non-terminal symbols, Σ the set of terminal symbols, R the set of derivation rules and S the start symbol, cf. Jurafsky and Martin (2021).
A small grammar producing the derivation of the example sentence consists of:

$N = \{S, NP, VP, DT, NN, JJ, VBG, VBZ\}$
$\Sigma = \{a, ball, cat, chasing, is, large\}$
$R = \{$
$S \rightarrow NP\ VP$
$NP \rightarrow DT\ NN \mid DT\ JJ\ NN$
$VP \rightarrow VBG\ NP \mid VBZ\ VP$
$DT \rightarrow a \mid the$
$NN \rightarrow ball \mid cat$
$JJ \rightarrow large$
$VBZ \rightarrow is$
$VBG \rightarrow chasing$
$\}$
$S = S$

Each sentence that can be derived with the rules of this grammar is said to be grammatical. This grammar creates tree structures as shown in fig. 1 with constituents as non-leaf nodes and words as leaf-nodes. It is also possible for rules to be recursive. For instance the VP consisting of a verb 3rd person sing. present (VBZ), *is* and another VP. Obviously, this is only a very small subset of the real CFG used to model the English language but hopefully sufficient to explain the main concepts.

2.2 Dependency Grammar

As an alternative formalization of syntactic structure, dependency grammar has emerged in European linguistics during the twentieth century and is especially popular in NLP systems of recent years (de Marneffe and Nivre (2019)). One of the first to formalize a theory of dependency grammar was Tesnière (1959) with his book *Élements de syntaxe structurale*. In dependency grammar, there are no constituents. Instead, there are only dependency relations between the words, modelling the syntactic structure of the sentence. A dependency relation consists of a head word, its dependant and the syntactic relationship between them.
The dependency tree of the example sentence *"the large cat is chasing a ball"* is shown in figure 2. All dependants, their head and the syntactic relation are also listed in table 2. There are no constituent nodes in the tree. Instead, the verb *chasing* can be interpreted as the root head node, with its dependants *cat* as the subject noun (nsubj), *is* as auxiliary verb (aux) and *ball* as its direct object (dobj). The noun *cat* in turn is the head of its dependant determiner *the* (det) and modifying adjective *large* (amod). The noun *ball* is the head of its determiner *a*.

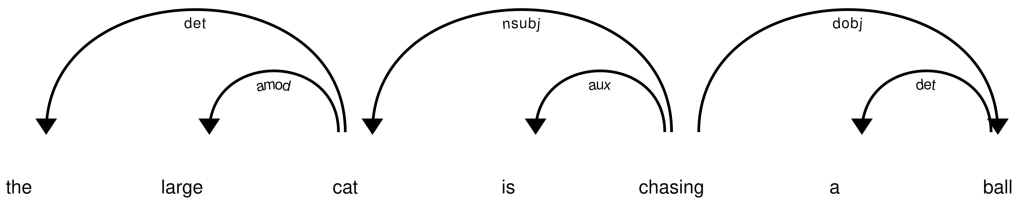


Figure 2: Dependency tree for 'the large cat is chasing a ball' - generated by spaCy parser (Honnibal and Montani, 2017) and their visualization tool displaCy (Honnibal and Montani, 2022)

Dependant	Head	Relation
the	cat	det
large	cat	amod
cat	chasing	nsubj
is	chasing	aux
chasing	chasing	root
a	ball	det
ball	chasing	dobj

Table 2: Sentence dependencies

2.3 Constituency and Dependency Grammar in NLP

Constituency and dependency grammar-style trees can both be used to model syntactic structure in Natural Language Processing systems. However, each have their own advantages and drawbacks which will be briefly discussed here. For more details, cf. Jurafsky and Martin (2021) and de Marneffe and Nivre (2019).
Compared to constituency grammar, dependency grammars have been gaining in popularity since they possess some characteristics which can be advantageous for NLP systems:

- They produce a spanning tree of asymmetric binary relations over the words of a sentence. This limits the number of nodes in the tree to the number of words, which makes them more compact, efficient and easier to parse. The parser only has to produce relations between existing nodes, not infer new constituent nodes as with constituency-style trees.
- They directly encode syntactic relationships between words, e.g. that *cat* is the subject and *ball* the object of the verb *chase*. This makes them useful for relation extraction tasks, such as *X lives in Y* or question-answering.
- They have a flexible word order. The order is determined by the relations between the words. Thus, dependency trees are able to abstract over differences in word order in different languages, making them useful for cross-linguistic tasks, e.g. translation.

Of course, these advantages also come with drawbacks:

- Because there are no constituent nodes, some relationships are difficult to represent in dependency grammar. For example, it is not possible to distinguish scope. Consider the phrase *young men and women*. Here, *young* modifies *men and women* but this is not expressed in dependency grammar.
- Because of their limitation to syntactic structure, dependency trees fail to represent non-syntactic phenomena as accurately as constituency trees. For example, Part-of-Speech (POS) tags such as the information that a word such as *John* or *New York* are proper nouns (PPN) do not exist in dependency trees.
- Because word order is not preserved, it is impossible to restore the original sentence from the dependency tree. Thus, the word-to-nodes mapping needs to be extracted and stored during pre-processing if it is needed further down the line.

So, in brief, constituency trees can sometimes be more expressive because of their notion of constituency, which introduces the ability to define scopes, as well as their native use of Part of Speech tags. Dependency grammars on the other hand, can be useful for tasks where the relationships between the words are of great importance.

3 Technical Groundwork

Discussion of technical basics used in the model. The basics of transformer architecture are reviewed, as well as BERT and graph neural networks.

3.1 Transformer

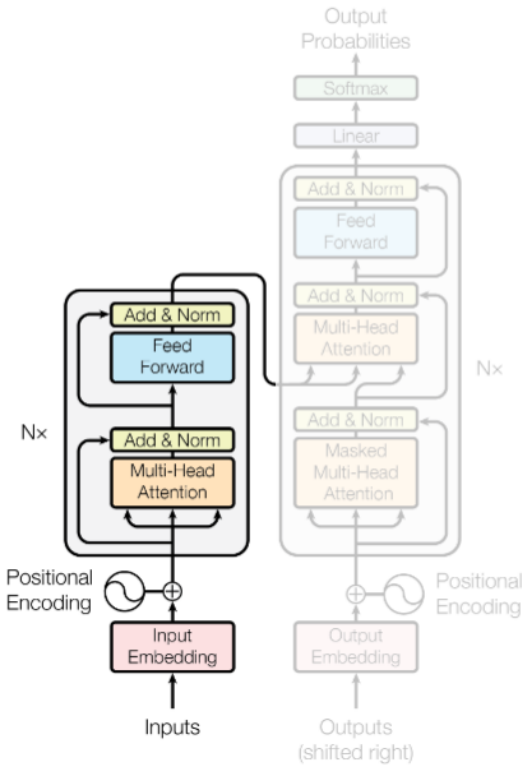


Figure 3: Transformer architecture overview - adapted with permission from Vaswani et al. (2017)

Transformers were first introduced in the now seminal paper *Attention is All You Need*, (Vaswani et al. (2017)). Figure 3 shows an overview of the transformer structure. It is an encoder-decoder model working on sequences and especially suited for tasks such as language translation. The encoder receives as inputs a sequence of input embeddings along with their positional encoding - for translation this might be a string of tokens in the input language. The Multi-Head Attention layer calculates the attention values of each token with all other tokens in the sentence (N times). The output is then passed through a Feed-Forward layer and sent to the decoder. Each layer also has a residual connection with its input.

The decoder will be only briefly discussed here, because in our model architecture, only the encoder is used. During training, the decoder receives the *labels*, a reference sequence in the target language. Its output are softmax values indicating the probability for a token in the vocabulary to be the next token in the target language. The first Masked Multi-Head Attention layer calculates important relationships between the words of the target sequence. To avoid the model ”looking ahead” in the sequence, all tokens after the current one are set to 0 in the attention mask. The second attention layer calculates important relationships between the words of the input and target sequence.

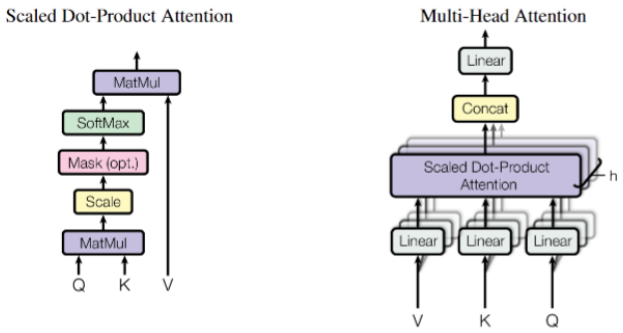


Figure 4: Transformer multi-head attention - reproduced with permission from Vaswani et al. (2017)

The Multi-Head Attention (see figure 4) consists of multiple concatenated Scaled Dot-Product attention layers. They receive Queries Q, Keys K and Values V, which are created by a linear transformation of the input embedding with separate trainable weights $W_{Q/K/V}$. The attention values are calculated as follows:
$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V$
In this way, each attention head can attend to and represent different token relationships.

3.1.1 BERT

BERT (Devlin et al. (2018)) stands for *Bidirectional Encoder Representation for Transformers* and is essentially a transformer, pre-trained on special NLP tasks using big corpora. It is based on multiple transformer encoders with multi-head attention.
The pre-training tasks consist of:

Masked Language Model: the model is trained with 15% of all tokens replaced. The original tokens are used as labels and the task is to correctly restore them. Of the replaced tokens, 80% are replaced with a masking token [MASK], 10% are replaced with a random token and 10% are kept. This helps to avoid that BERT only learns context information for masked tokens and always predicts the input token for every non-masked token. For this, two special tokens are needed: [CLS] marking the first sentence and [SEP] marking the beginning of the second sentence.

Next Sentence Prediction: the task is to predict whether two sentences are likely to follow each other or not.
The labels for both tasks can be generated automatically, so pre-training is fast. The pre-training tasks are generic enough so that they are a useful basis for a variety of other NLP tasks such as sentiment analysis, question answering or summarization of texts.
BERT uses WordPiece embeddings of size 768 as internal representations of words. For this, there exists an internal vocabulary of tokens of size 30522. The token ids are then mapped onto the embeddings during training/inference. The complete input then consists of the token ids padded to a fixed sequence length, mask ids to indicate tokens to ignore during training, segment ids to distinguish the two sentences used for the Next Sentence Prediction task and positional embeddings, indicating the token position in the sentence.
For example, the sentence *The flight to Nanterre goes today* and a sequence length of 12 would be tokenized as follows:

Position id	0	1	2	3	4	5	6	7	8	9	10	11
Token	[CLS]	the	flight	to	nan	##ter	##re	goes	today	[SEP]	[PAD]	[PAD]
Token id	101	1996	3462	2000	16660	3334	2890	3632	2651	102	0	0
Mask id	1	1	1	1	1	1	1	1	1	1	0	0
Segment id	1	1	1	1	1	1	1	1	1	1	1	1

Table 3: BERT tokenization example

The embedding for a token is then given by the token embedding E_{token_id} plus the segment embedding $E_{segment\_id}$ and its positional embedding $E_{position\_id}$. Words that are not in the vocabulary are split into the smallest fragments still included in the vocabulary. This avoids large amounts of words characterized as e.g. the same [UNK] token.

3.2 Graph Neural Networks

Graph Neural Networks (GNNs) are able to process graph structures and extract information from the structured data not available in other networks such as plain Convolutional Networks. A graph is a data structure consisting of nodes and edges between the nodes. Zhou et al. (2020) distinguish two main types of graph convolutional networks. Spatial graph convolution networks working with the spatial structure of the graph and evaluating neighbourhood relationships between the nodes. The second type are spectral graph convolutional networks transforming the graph to the spectral domain and working with this representation.
This introduction will focus on spatial graph convolutional networks, a subtype of which is also used later in the model architecture.

3.2.1 Message Passing

Spatial convolutional networks work with the concept of *message-passing* between neighbouring nodes. The graph convolution consists of aggregating the features of all neighbouring nodes into the central node and summing the results. In this way, all neighbours influence a node's features which is helpful to discover relationships between them, e.g. nodes that often occur together.
The drawback of this approach is that there is no way to distinguish between the importance of a single neighbour, they all have the same impact. If there are a lot of neighbours, this can mask important features by unwanted noise.

3.2.2 Graph Attention

A solution to this problem is a variation of the spatial graph convolutional network using attention. It works by introducing attention weights for the neighbouring nodes, thus distinguishing important signals from noise. It is also possible to use it with multi-head attention (Vaswani et al. (2017)). Figure 5 shows the structure of a multi-head graph attention network (GAT). First, all node embeddings h_i are sent through a linear layer with weights vector W, producing x_i.

$$x_i = Wh_i \tag{1}$$

For all pairs of nodes x_i and $x_j \in \mathcal{N}(i)$ with $\mathcal{N}(i)$ being the set of all 1-hop neighbours of x_i, attention coefficients $\alpha_{i,j}$ are calculated. It consists of the dot product of the attention

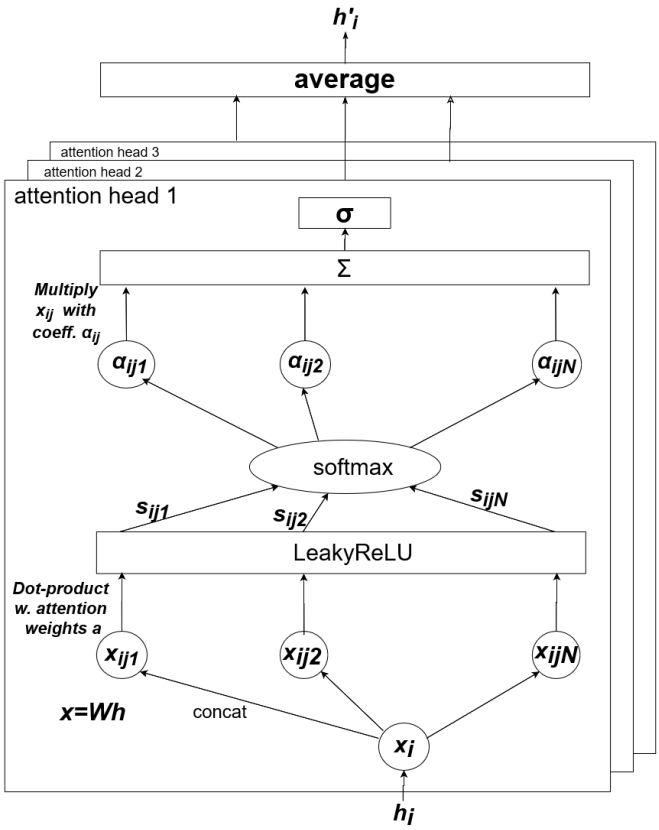


Figure 5: Multi-head graph attention

weights vector a and the concatenated node features of x_i and x_j, concatenation being represented as $||$. This is then sent through a LeakyReLU (Xu et al. (2015)) activation function, producing $s_{i,j}$.

$$s_{ij} = \text{LeakyReLU}(\vec{a}^{\top}(x_i||x_j)) \tag{2}$$

To scale the results, softmax is performed to create the final attention coefficient.

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{k\in\mathcal{N}(i)} \exp(s_{ik})} \tag{3}$$

Thus, the features of all neighbouring nodes are scaled by their attention coefficient $\alpha_{i,j}$, attenuating some and increasing others. These scaled features of all neighbouring nodes are then summed and sent through an activation function σ.

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)} x_j^{(l)}\right) \tag{4}$$

This process can be performed by K heads in multi-head attention, where each head has its own weights trained separately and is 'attending' to different features. The results of each head are then either concatenated or averaged over for the final layer output.

3.3 Applications

Graph neural networks are used in a variety of applications. In biology and medical engineering they help to predict protein folding mechanisms or the prediction of chemical reactions. They can be used for recommendation systems in social networks or when recommending items to a potential customer. In a field closer to NLP, GNNs can efficiently leverage knowledge graphs, collection of facts about real-world entities and the relationships between those entities. Knowledge graphs are used in question answering, information retrieval and other fields. Examples taken from (Zhou et al. (2020)).

3.4 Libraries and Frameworks

In this section, the libraries and frameworks used to implement the model are described. It was programmed in Python 3 Van Rossum and Drake (2009) and for the general model architecture, *PyTorch* (Paszke et al. (2019)) is used. PyTorch is an open-source deep-learning library for Python. It comes with its own datatype called *Tensors* which enable efficient gradient computation by storing a history of the operations performed on them. Also, PyTorch offers many pre-defined network layers and functions.
It is also necessary to have baseline BERT models, which are extended by the GNN architecture described in this work. For this, the implementations available from Hugging Face are used. Specifically, the models *bert-base-cased* (Hugging Face (2022a)), which is trained with a tokenizer distinguishing between upper- and lower-case words and *bert-base-uncased* (Hugging Face (2022b)), converting everything to lower-case.
The Graph Neural Network is implemented with the library *PyTorch Geometric* (PyG) (Fey and Lenssen (2019)). PyG includes implementations for GNN architectures from various publications and papers. It also provides a consolidated data structure for representing graphs and feeding them to the network. This data structure consists of the list of node features *x* of size [num_nodes, num_node_features], the list of edges *edge_index* of size [2, num_edges] represented by their source and target nodes and an optional list of edge features *edge_attr* of size [num_edges, num_edge_features].

4 Related Work

The idea to enhance NLP models such as BERT by enriching them with syntactic information is not new and has been around for a few years. However, the methods employed to do this have evolved a fair bit over time.

Sundararaman et al. (2019) modify the BERT WordPiece embeddings to include only the Part-of-Speech (POS) information for each token. This enables the model to distinguish e.g. between a noun and a verb, but does not take into account the syntactic relationships between the words.

Other approaches utilize the information available from syntax graphs to various degrees. The authors of LISA (Linguistically-Informed Self-Attention), Strubell et al. (2018), incorporate the syntax graph structure by training one attention head to attend to the syntactic parent of each token. However, other information such as the siblings of a token or the nature of the relationship are lost. Bai et al. (2021) improve on this approach by introducing separate attention masks not only for the parent, but also the children and the siblings of a token. They use both constituency and dependency trees to create their masks. The SG-Net model (Zhang et al. (2020)) does something similar, but employs only one mask for each token, determining that it should only attend to its direct neighbours in the syntax tree.

The latest stage of development is to incorporate syntax graphs by leveraging GNNs for this purpose. Goel and Sharma (2022) use an LSTM coupled with a Graph Convolutional Network (GCN) processing dependency trees to detect offensive language in social media messages. With this model, they outperform BERT in the detection task. They also briefly test the effect of using a GCN layer on top of BERT but see detrimental results.

Sachan et al. (2021) describe two different methods to combine a BERT encoder with a GNN processing dependency graphs. Either by incorporating the GNN output in the BERT embeddings (“*Joint Fusion*”) or by placing the GNN on top of BERT (“*Late Fusion*”) and combining their outputs. Their last method achieves improved results in a variety of NLP tasks compared to the original BERT. The model built in this work is also in large parts modelled after the Late Fusion architecture described in their publication.

Dataset	# Sentences	Description
Atis - Airline Travel Informations	5432	speech transcripts of flight requests
ESL - English as a Second Language	5142	Cambridge FCE exams
EWT - English Web Treebank	16621	weblogs, newsgroups, emails, reviews
GUM - Georgetown University Multilayer	9130	multiple text types
LinES - Linköping English-Swedish Corpus	5243	parallel database of English and Swedish
PUD - Parallel Universal Dependencies	1000	news and Wikipedia entries
ParTUT - Turin University Parallel Treebank	2090	written texts
Total	46,553	$\sim$714k words

Table 4: Universal Dependencies English datasets

5 Data

5.1 Universal Dependencies

Universal Dependencies is a project committed to delivering consistent dependency grammar annotations across different languages (Nivre et al. (2020)). The tagset for dependency relations is a modified version of the Stanford dependency tagset comprising 42 tags, with the possibility of adding language-specific tags if needed (de Marneffe et al. (2014)). Their hand-annotated data for the English language is processed and then used as input data for the model training with dependency-grammar style syntax trees. All available datasets are listed in table 4.
The data is given in CoNLL-U format (Marneffe et al. (2014)), a modified form of the CoNLL-X format (Buchholz and Marsi (2006)) introduced by the Conference on Computational Natural Language Learning. It consists of the following fields describing the syntax trees (* = optional):

1. ID: Word index
2. FORM: Word form or punctuation symbol.
3. LEMMA: Lemma or stem of word form.
4. UPOS: Universal part-of-speech tag.
5. XPOS: Language-specific part-of-speech tag (*)
6. FEATS: List of morphological features from the universal feature inventory (*).
7. HEAD: Head of the current word, ID or zero.
8. DEPREL: Universal dependency relation to the Head.
9. DEPS: list of head-deprel pairs.
10. MISC: Any other annotation.

5.2 International Corpus of English (ICE-GB)

The International Corpus of English (ICE) is a collection of texts from a variety of English dialects. The corpus for Great Britain (ICE-GB) also offers hand-annotated constituency tree information (Nelson et al. (2002)). ICE-GB was compiled by the Survey of English Usage at the University College London (UCL Survey of English Usage (2020)). This data is processed and then used as input data for the model training with constituency-style syntax trees. Table 5 shows the collection of datasets used from the corpus. Printed or written data was preferred.

Dataset	# Sentences	Description
printed text	20230	books, journals, magazines, newspapers
written letters	4763	written social and business letters
written student essays	2471	student essays end exams
spoken scripted text	3401	scripted lectures and talks
Total	30865	

Table 5: International Corpus of English (GB) datasets

The data is available as text files, the example in 1 shows the representation of the syntax tree for the sentence *How are the wonderful people at Nanterre?*. Each line represents a node, with the indentations denoting its level in the tree. For example, the line *AVHD,ADV(wh) {How}* indicates that the word *How* has the part-of-speech tag *ADV* and is the head (*AVHD*) of an adverbial phrase. It also includes other attributes in brackets, in this case, it is the information that it is classified as a question word *wh*.

```
PU,CL(main,inter,cop,pres,precs)
 CS,AVP(wh)
  AVHD,ADV(wh) {How}
 INTOP,V(cop,pres) {are}
 SU,NP
  DT,DTP
   DTCE,ART(def) {the}
  NPPR,AJP(attru)
   AJHD,ADJ(ge) {wonderful}
  NPHD,N(com,plu) {people}
  NPPO,PP
   P,PREP(ge) {at}
   PC,NP
    NPHD,N(prop,sing) {Nanterre}
PUNC,PUNC(qm) {?}
```

Listing 1: Example of a constituency tree file

5.2.1 spaCy and Berkeley Neural Parser

In addition to the hand-annotated syntax trees for the data, machine-generated syntax trees for the ICE-GB data were also created. This enables us to compare the model performance with machine-generated and hand-annotated syntax trees. The syntax trees were generated with the NLP framework spaCy (Honnibal and Montani (2017)) and the Berkeley Neural Parser (Benepar), a constituency parser available as a plug-in for spaCy. The Benepar model consists of a span-based encoder with self-attention and received high F1 scores (95.40) on the English News Text Treebank (Kitaev and Klein (2018), Kitaev et al. (2019), Linguistic Data Consortium (2022)).

5.3 Data Preprocessing

The original files containing the syntax graphs are converted to Pytorch Geometric Data objects to be used as input for the GNN. With the constituency graphs of the ICE-GB dataset, there are two different types of nodes: the constituent nodes and the leaf nodes containing the words of the sentence.
The words of the sentence are represented by their respective Bert embeddings of size 768. Since the Bert WordPiece tokenizer splits words into separate tokens, the subtokens of a word are added to it as child nodes. The constituent labels are one-hot-

encoded and represented as an all-zero vector of the same size as the Bert embeddings with only one value set to 1.
The constituency graph has no edge attributes and starts with the root node *CL* or *NONCL*. These root nodes are equivalent to the root node *S* used in the introductory chapter about constituency grammar. Figure 6 shows the constituency tree for the sentence *How are the wonderful people at Nanterre?*. Here, *Nanterre* is split into separate tokens, with the subtokens being attached to the first.

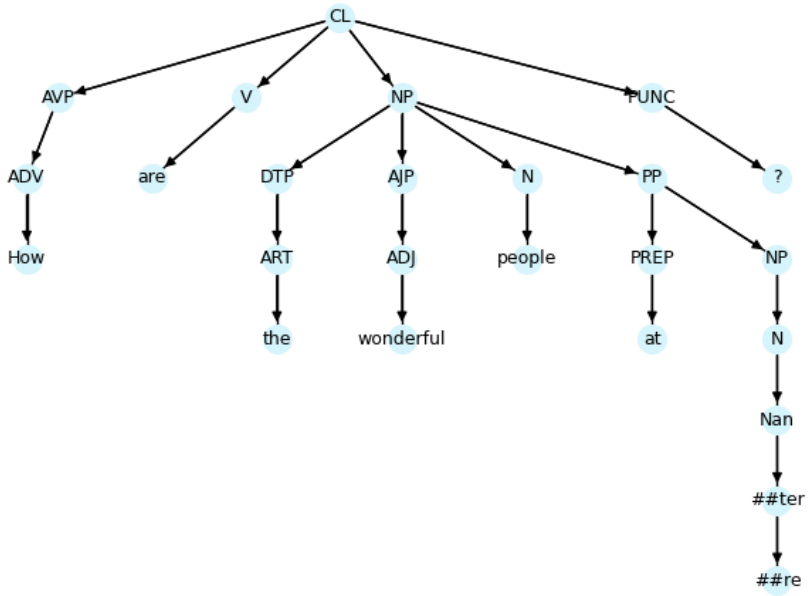


Figure 6: Pytorch geometric constituency tree representation

The dependency graphs are modelled similarly. The nodes are the tokens of the sentence and the dependency relations are modelled as edge attributes. A new relation, *sub*, is used to mark the connection between a word and its subtokens. Figure 7 shows the Pytorch Geometric representation of the dependency tree for *Sociologists have explored the adverse consequences of discrimination[3-5];*. It is clear, that it is very hard to map the original sentence onto the graph structure. Since we will need this information to replace the Bert embeddings with updated ones during training, a mapping is stored along with each Pytorch Geometric graph. It maps the index of a word in the sentence to its position in the graph.
Sometimes, this mapping is impossible to do without detailed dissection of both graph and sentence. This might be the case, if the the sentence contains words which are split up in the graph, e.g. *don't* as *do n't*. Because of time constraints it was decided to

remove sentences with an incomplete mapping. This leads to a final count of correct graphs and sentences of 41,222 used as model input.

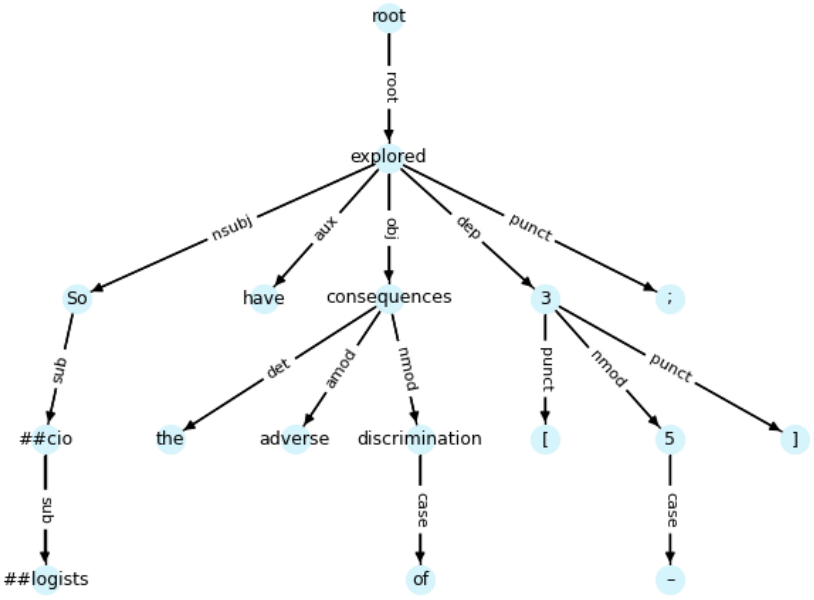


Figure 7: Pytorch geometric dependency tree representation

All datasets were split up into train, validation and test data with a ratio of 75% test data, 12.5% validation data and 12.5% test data.

5.4 Named Entity Labels

The main task used for evaluating the model performance is Named Entity Recognition. Since named entity labels are available neither for the UD nor the ICE-GB dataset, the NER labels are created automatically with Flair, a NLP framework including several Bert-based pre-trained Named Entity Recognition models. (Akbik et al. (2019)). Specifically, the *ner-english-ontonotes-large* model is used for tagging. It was trained on the Ontonotes data and contains 18 different NER tags, listed in table 6.
The NER model uses the BIOES system to specify NER tags spanning multiple words, see table 7 for an overview. These are prepended to an NER tag to show the position of a word inside of an Named Entity, or as in the case of the O, used alone to indicate that a word is not a Named Entity. For example, the sentence *John W. Smith is turning thirty* would be tagged as *[B-PERSON, I-PERSON, E-PERSON, O, O, S-CARDINAL]*.

NER Tag	Description	Example
CARDINAL	cardinal value	thirty
DATE	date value	December 1st
EVENT	event name	Easter, Fan Expo Canada
FAC	building name	the White House, Johnson Space Center
GPE	geo-political entity	France, Germany
LANGUAGE	language name	French, German
LAW	law name	Creative Commons, Open Source GPL
LOC	location name	West Coast, the South
MONEY	money name	1 dollar, US$5
NORP	national or religious affiliation	Jewish, Muslim, French
ORDINAL	ordinal value	first, 33rd
ORG	organization name	the Red Cross, SETI
PERCENT	percentage	85%
PERSON	person name	John W. Smith
PRODUCT	product name	PlayStation VR, Apollo [spacecraft]
QUANTITY	quantity value	2kg, 20 square meters
TIME	time value	6:30pm in the afternoon, yesterday
WORK_OF_ART	name of work of art	Game of Thrones

Table 6: Named Entity Tags predicted by Flair model

Prefix	Description
B	first word of an NE
I	word inside of an NE
O	word is not part of an NE
E	word occurs at the end of an NE
S	word is a complete/single NE

Table 7: BIOES tags in Named Entity Recognition

Since BERT splits words into tokens, we also need a way to deal with those subword tokens. Different strategies exist, but the one adopted here is to introduce another NE tag, *X* with which all subword tokens as well as padding tokens are tagged. Another possibility is to give all subword tokens the same NE tag as the original word, but this falsifies the information given to the model, which might then learn that the subtokens constitute separate entities also tagged with these NE tags.

6 Model Architecture

6.1 Overview

The model architecture is based on an architecture described in the paper by Sachan et al. (2021). It consists of the encoder part of BERT, a GNN processing the syntax graphs and a Highway Gate. The authors refer to this architecture as *Late Fusion*, because the syntax-GNN (SynGNN) is stacked onto the original BERT encoder. The encoder outputs its raw hidden states, i.e. embeddings, which are fed to the SynGNN along with the syntax graphs. In the paper it is left unclear how exactly the embeddings are included in the syntax graphs. The most probable approach is to embed them into the syntax graphs as the token node features, so this is the method that was adopted. With each forward pass, the updated BERT embeddings are integrated into the syntax graph before passing it into the SynGNN.
Next, the output of the SynGNN and the original BERT embedding output are routed through a Highway Gate (Srivastava et al. (2015)). This enables the model to select the most useful representations from both BERT and the SynGNN.
Finally, there is a task-specific classifier layer. For the NER task, the classifier layer is of the same size as the number of possible NER labels. Figure 8 shows a block diagram of the model structure.

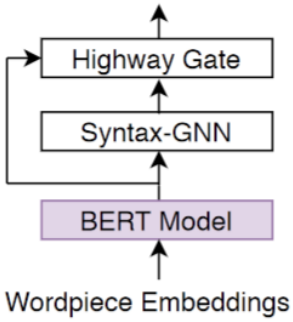


Figure 8: Model architecture overview - weights shown in color are pretrained, uncolored weights are initialized with Xavier uniform initialization by Glorot and Bengio (2010). Reproduced with permission from Sachan et al. (2021)

6.2 SynGNN

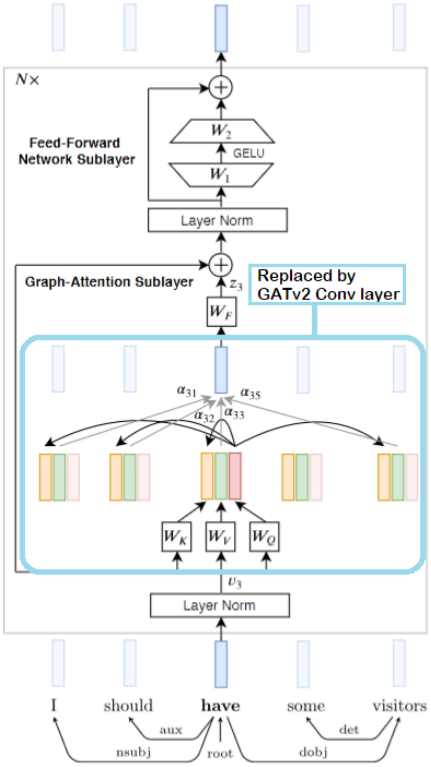


Figure 9: SynGNN layer overview - reproduced and adapted with permission from Sachan et al. (2021)

The SynGNN is a variation of the transformer encoder where the self-attention layer is replaced with a graph attention layer. It consists of multiple layers - the construction of a single SynGNN layer is shown in figure 9. Each layer contains a Graph Attention submodule and a Fast-Forward Network submodule. Its construction is described in detail in Sachan et al. (2021) and summarized here. The input to a SynGNN consists of a syntax graph, either of dependency- or constituency-style, including the BERT Wordpiece embeddings in the token nodes. The input data is normalized and then fed into the graph-attention sub-module. The input to the graph attention submodule consists of the input node embeddings $V = \{v_i \in \mathbb{R}^d\}_{i=1:\mathcal{N}^v}$ and the edges of the syntax tree $E = \{(e_k, i, j)_{k=1:\mathcal{N}^e}\}$ with e_k the edge from node i to node j and the edge attributes. The authors of the original paper employ the GAT mechanism as described in the section *Technical Details* (3.2.2), which only uses the node features in the calculations. Since the dependency trees also have edge features which should not be ignored, a Pytorch Geometric GATv2Conv layer was chosen for this model instead. The layer

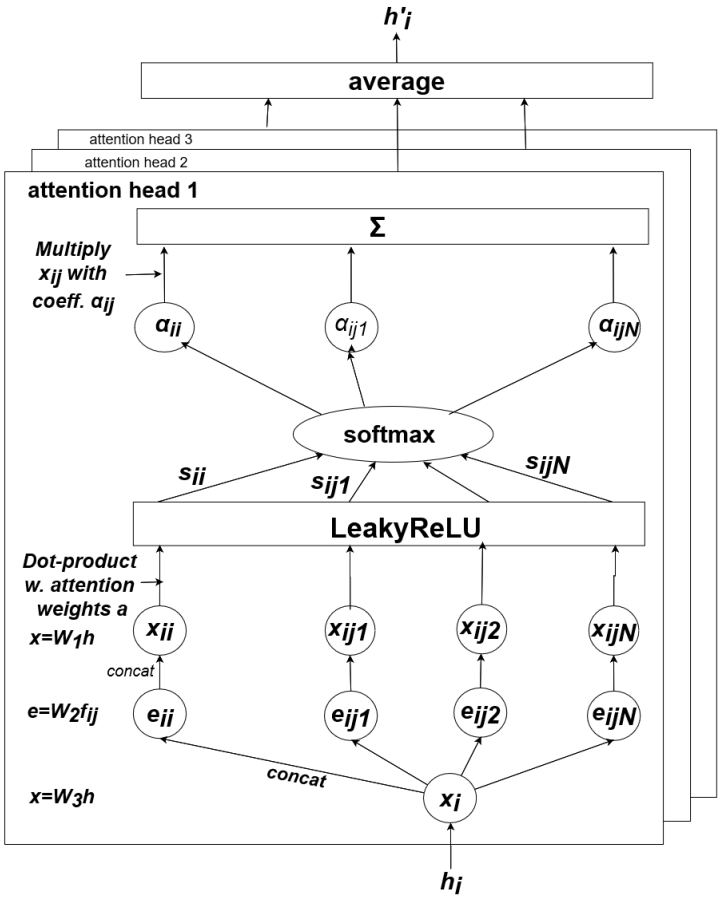


Figure 10: Pytorch Geometric GATv2 Conv layer architecture

architecture is shown in figure 10. This layer is based on the paper Brody et al. (2021) and also includes the edge features when updating the node features with the attention scores. A node x_i is not only concatenated with the node features of its neighboring node ij but also with the linearly transformed edge features $e_{i,j}$ when calculating the attention scores $\alpha_{i,j}$. The GATv2Conv layer also includes three separate linear layers with separate weights to produce x_i, x_j and $e_{i,j}$ instead of only one shared weight. Also, self-loops are introduced for each node during the calculation, so that the node pair x_i x_i will also be processed during the calculation of the attention scores.
The equation given for the layer output including the self-loop in the Pytorch Geometric documentation is:

$$\mathbf{h}_i' = \alpha_{i,i}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\mathbf{x}_j \tag{5}$$

Note also, that the second activation function present in the original GAT architecture is missing here in the GATv2 Conv layer.

$\alpha_{i,j}$ including the edge features is calculated as:

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{W}_1\mathbf{x}_i \parallel \mathbf{W}_3\mathbf{x}_j \parallel \mathbf{W}_2\mathbf{e}_{i,j}\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{W}_1\mathbf{x}_i \parallel \mathbf{W}_3\mathbf{x}_k \parallel \mathbf{W}_2\mathbf{e}_{i,k}\right)\right)} \tag{6}$$

with $\mathcal{N}(i) \cup \{i\}$ being the set of node features of x_i's neighbours including itself, i.e. the self-loop and $e_{i,j}$ the node features.

Referring back to the SynGNN layer architecture in figure 9, the graph attention output z_i is then combined with the original layer input through a residual connection. The result is normalized and passed through the fast-forward network consisting of two linear layers and a GELU (Gaussian Error Linear Unit) activation function (Hendrycks and Gimpel (2016)). Here again, the sublayer output is combined with its input via a residual connection for the final output.

$$\text{FFN}(z_i) = \text{GELU}(z_i\mathbf{W_1} + b_1)\mathbf{W_2} + b_2 \tag{7}$$

In equation 7, W_1 and W_2 refer to the weights of the linear layers of the Feed-Forward Network sublayer of the SynGNN layer. The SynGNN combines multiple SynGNN layers by stacking the outputs onto each other.

6.2.1 Highway Gate

Finally, after all SynGNN layers have been calculated, the SynGNN produces its final output. This output is combined with the original BERT output embeddings via a Highway Gate as described by Srivastava et al. (2015) and Sachan et al. (2021). Let v_i be the BERT output and z_i be the SynGNN output. Then the final output h_i is computed as:

$$g_i = \sigma(\mathbf{W_g}v_i + b_g) \tag{8}$$

$$h_i = g_i \odot v_i + (1 - g_i) \odot z_i \tag{9}$$

with g_i a linear layer with the BERT output as input and a sigmoid activation function. h_i is then calculated as the Hadamard product ($\odot$) of g_i and its original input v_i, the BERT embeddings, combined with the Hadamard product of the inverse of g_i and the SynGNN output z_i.
Finally, the output of the highway gate is routed through a classifier layer. With the NER task this classifier layer outputs a value for each of the different NE tags.

6.3 Label Weights

Label weights are needed to adjust the loss function in the case of imbalanced datasets. With the NER task, we do have exactly such a case. Recall that the *O* label is used to label any token that is not a NE. This means that this label is highly over-represented in the dataset and thus much more likely to be chosen by the model to optimize its loss. To combat this behavior, the label weights are adjusted according to their frequency in the data. The method used is re-implemented from and based on the ***balanced*** heuristic used in the scikit-learn function ***compute_label_weights*** (scikit-learn (2022)).
The heuristic is calculated as $n\_samples/(n\_classes * np.bincount(y))$ where $n\_samples$ is the number of samples in the training data, $n\_classes$ the number of unique classes and y the true labels of the train data. The ***bincount*** function returns an array of occurrence counts for all unique labels.
Additionally, a ***clip_value*** parameter is introduced, which is not present in the original weights calculation function. The resulting label weights are clipped when they are higher than a specified value, e.g. 50. Its purpose is to prevent an "overshoot" where very rare NE labels are boosted too much and are predicted more often than they should be.
The original authors of the paper on which the model architecture is based do not mention any adjusting for the imbalanced nature of the NE labels. As it was found that the model did not perform well without the label weights, however, they were integrated as default in the model created here.

6.4 Metrics

To train and evaluate the model, several metrics are used which are introduced here. As loss function for model training ***Cross-Entropy Loss*** is employed, using the implementation available in PyTorch (cro (2022)). According to the documentation, the Cross-Entropy loss for a Tensor x of size $(batch\_size, C)$ containing the predicted scores for each class and a Tensor y of size $(batch\_size, 1)$ containing the true labels is calculated as follows:

$$\begin{aligned} &\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \\ &l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\} \end{aligned} \tag{10}$$

with x the input, y the target labels, C the number of label classes and N running over the batch size. For each (x_n) first the softmax is calculated over all scores, followed by the cross-entropy loss.
Optionally, one can specify *ignore_index* to ignore a specific target label in the calculations. This is used to ignore all subtokens of a word as well as padding tokens, which are both tagged with the special target label *X*. In this way, one word still corresponds to exactly one label as the subtokens are removed from the calculation.
Also, it is possible to modify the loss with a weight w_{y_n} corresponding to the target label y_n. For this, a Tensor with desired weights for each label class, as calculated in the previous section, is be given to the function. Otherwise, all label weights are set to 1. All cross-entropy loss values across the batch are then combined to the batch loss by calculating their mean:

$$\ell(x, y) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot 1\{y_n \neq \text{ignore\_index}\}} l_n \tag{11}$$

To assess model performance, metrics originating in information retrieval, *Precision*, *Recall* and *F1* (see Russell and Norvig (2016)) are employed. *Precision* measures the proportion of all retrieved items that are actually relevant. Or, in the case of our classification task, the proportion of items that were correctly classified as belonging to a class among all that were classified as that class, including incorrect classifications. This measures the ability of the model to distinguish members of a class from non-members , with 1 being the highest and 0 the lowest possible value. The two groups are called True Positives (TP) and False Positives (FP). Conversely, True Negatives (TN) refers to items that were correctly classified as not belonging to a class, whereas False Negatives (FN) are items incorrectly classified as not belonging to a class while in truth they do. Thus, *precision* is defined as:

$$P = \frac{TP}{TP + FP} \tag{12}$$

Recall measures the proportion of all relevant documents that were retrieved. In our classification task, this refers to the proportion of items that were correctly classified as being in a class, to all classifications of members of that class, including those wrongly classified as something else. This measures the ability to correctly identify all members of a given class, with 1 being the highest and 0 the lowest possible value. *Recall* is defined as:

$$R = \frac{TP}{TP + FN} \tag{13}$$

F1 then is the harmonic mean of precision and recall:

$$F1 = 2PR/(P + R) \tag{14}$$

Specifically, the Python library *seqeval* (Nakayama (2018)) is used, which was designed for the evaluation of sequence labelling tasks such as NE or Part-of-Speech (POS) tagging. Here, a NE tag counts as correctly identified only if all of its parts have been correctly classified. For example, the sequence *John [B-PERSON], W. [I-PERSON], Smith [E-PERSON]* would count as a single, correctly identified *PERSON* tag only if the model predicted [B-PERSON, I-PERSON, E-PERSON] and as not correct in all other cases.

The library also calculates the average of a given metric over all NE labels. The *micro average* calculates TP, FP and FN over all label classes and then calculates precision, recall and F1 values from those total counts. The *weighted average* is obtained by calculating precision, recall and F1 over each class separately weighted by the number of true instances, summing over them and dividing by the total number of instances. Since recall is only calculated using members of one class, the micro and weighted average recall are the same value.

Before calculating the metrics, the sequence tokens are pruned similarly to the loss calculation. All subtokens of words are ignored, as well as padding tokens and the *[CLS]*, and *[SEP]* tokens BERT uses at the beginning and end of the sequence.

7 Experiments

7.1 Overview

To evaluate the performance of the SynGNN, a number of experiments were conducted. First, a baseline model was chosen to compare the SynGNN against. Next, the SynGNN is trained with the hand-annotated, gold-standard syntax graphs. Here, the influence of the label weights clipping parameter is given special attention. To compare the influence of the gold standad syntax graphs to the generated ones, the experiments are repeated with generated syntax graphs. Lastly, since the dataset sizes of the dependency and constituency dataset are not the same, another experimental run is done where the dependency dataset is trimmed to contain the same amount of data as the constituency dataset. This is done to ensure a fair comparison between the performance of dependency and constituency graphs.
To answer the question whether the SynGNN really profits from the information the syntax graphs can provide, and not only other factors such as the use of label weights, the BERT baseline model is also tested with those enhancements enabled.
The results for each of the experiments are presented in the following sections.

7.2 Baseline BERT Model

To evaluate the performance of the SynGNN model, a baseline to compare against is needed.
This baseline model consists of the original BERT encoder model combined with a classifier layer and a dropout. It is pre-trained on all available data of either the constituency tree (ICE-GB) or the dependency tree dataset (UD). The basic configuration is the following: pre-training for 10 epochs with all label weights set to 1, meaning no change in the loss function, and a learning rate of $2 * 10^{-05}$ with a linear decay to $0.3 * 2 * 10^{-05} = 6 * 10^{-06}$ during the first 5 epochs. This pre-trained model is also re-used as the standard BERT encoder in the SynGNN architecture.

7.2.1 BERT Cased

The main baseline model uses the *bert-base-cased* model which distinguishes between words in upper- and lower-case. The expectation is that this might help the model to achieve better results with the NER task, since a lot of Named Entities such as names of persons or places are typically spelt using upper-case first characters in English. The results for the NER task on the test data are given in table 8. The table shows precision, recall and F1 metrics using the text from the dependency and constituency datasets respectively. The metrics are given separately for each NE label class, along with the true number of instances the class occurs in the dataset. The last two rows show the micro average and weighted average over all classes. To compare model performance, the micro average will be used, since it gives more direct feedback on the overall performance and it is the metric customarily reported. In general, the BERT model trained on the dependency dataset is performing better than the one trained on the constituency data with a F1 score of 0.70 compared to 0.63. However, the dataset also contains a lot more Named Entities.

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.67	0.67	0.67	611	CARDINAL	0.65	0.72	0.68	528
DATE	0.66	0.73	0.69	1045	DATE	0.60	0.61	0.60	584
EVENT	0.49	0.51	0.50	80	EVENT	0.50	0.35	0.42	31
FAC	0.33	0.48	0.39	151	FAC	0.29	0.29	0.29	73
GPE	0.86	0.86	0.86	1936	GPE	0.74	0.69	0.72	416
LANGUAGE	0.59	0.30	0.40	77	LANGUAGE	0.67	0.55	0.60	11
LAW	0.57	0.67	0.61	57	LAW	0.27	0.27	0.27	22
LOC	0.52	0.51	0.51	217	LOC	0.50	0.55	0.53	130
MONEY	0.44	0.51	0.47	61	MONEY	0.46	0.46	0.46	24
NORP	0.69	0.77	0.73	422	NORP	0.76	0.73	0.75	247
ORDINAL	0.83	0.90	0.87	171	ORDINAL	0.74	0.69	0.72	84
ORG	0.64	0.63	0.64	857	ORG	0.59	0.62	0.61	469
PERCENT	0.49	0.64	0.55	36	PERCENT	0.43	0.58	0.50	43
PERSON	0.75	0.78	0.76	1370	PERSON	0.76	0.76	0.76	859
PRODUCT	0.38	0.45	0.41	98	PRODUCT	0.46	0.40	0.43	72
QUANTITY	0.29	0.38	0.33	53	QUANTITY	0.44	0.47	0.46	77
TIME	0.62	0.65	0.63	214	TIME	0.59	0.72	0.65	61
W._OF_ART	0.44	0.41	0.42	130	W._OF_ART	0.19	0.23	0.21	107
Micro Avg	0.68	0.73	0.70	7586	Micro Avg	0.62	0.65	0.63	3838
Weigh. Avg	0.70	0.73	0.72	7586	Weigh. Avg	0.64	0.65	0.64	3838

Table 8: NER results for BERT baseline model cased with dependency (UD) and constituency (ICE-GB) dataset

7.2.2 BERT Uncased

Table 9 shows the BERT baseline results when using the uncased model. Recall that the motivation for including this model was the assumption that it would probably perform more poorly than the cased model and thus potentially gain more from including additional information in the form of syntax graphs. Interestingly, this is not entirely the case. Looking at the F1 score, the model trained on the dependency data loses 1 point on its cased counterpart, landing at 0.69 and the one trained on the constituency data loses a more noticeable 3 points with a score of 0.60.

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.68	0.65	0.67	612	CARDINAL	0.73	0.61	0.67	528
DATE	0.69	0.72	0.70	1045	DATE	0.58	0.59	0.58	584
EVENT	0.40	0.44	0.42	80	EVENT	0.30	0.26	0.28	31
FAC	0.34	0.42	0.38	151	FAC	0.22	0.18	0.20	73
GPE	0.88	0.85	0.87	1936	GPE	0.70	0.67	0.68	417
LANGUAGE	0.49	0.34	0.40	77	LANGUAGE	0.78	0.64	0.70	11
LAW	0.41	0.46	0.43	57	LAW	0.16	0.14	0.15	22
LOC	0.50	0.50	0.50	217	LOC	0.56	0.51	0.53	130
MONEY	0.52	0.52	0.52	61	MONEY	0.32	0.25	0.28	24
NORP	0.68	0.75	0.71	422	NORP	0.75	0.66	0.70	247
ORDINAL	0.83	0.84	0.83	171	ORDINAL	0.78	0.68	0.73	84
ORG	0.55	0.57	0.56	857	ORG	0.50	0.54	0.52	469
PERCENT	0.55	0.64	0.59	36	PERCENT	0.41	0.44	0.43	43
PERSON	0.75	0.79	0.77	1371	PERSON	0.74	0.72	0.73	859
PRODUCT	0.29	0.43	0.35	98	PRODUCT	0.34	0.33	0.34	72
QUANTITY	0.35	0.42	0.38	53	QUANTITY	0.42	0.45	0.43	77
TIME	0.57	0.68	0.62	214	TIME	0.49	0.57	0.53	61
W._OF_ART	0.30	0.25	0.27	130	W._OF_ART	0.12	0.13	0.13	107
Micro Avg	0.67	0.71	0.69	7588	Micro Avg	0.60	0.59	0.60	3839
Weigh. Avg	0.70	0.71	0.70	7588	Weigh. Avg	0.62	0.59	0.60	3839

Table 9: NER results for BERT baseline model uncased with dependency (UD) and constituency (ICE-GB) dataset

7.3 SynGNN with Gold-standard Syntax Graphs

After establishing the baseline to compare against, it is now possible to compare the performance of the SynGNN model. The SynGNN is trained again with all available data and the hand-annotated, gold-standard syntax graphs. This section is presenting the results for the best SynGNN configuration found during the experiments. Specifically, using the pre-trained BERT baseline *bert-base-cased* as the internal BERT model, the model is trained for 15 epochs with a batch size of 32, calculated label weights as described in 6.3, a label weights clipping value of 50 and a a learning rate of *2e-05* with a linear decay to *0.2*2e-05 = 4e-06* over the first 8 epochs. The SynGNN model itself is configured with 4 layers and the *GATv2_Conv* layer has 2 attention heads. Unless otherwise mentioned, this is also the basic configuration used for any SynGNN discussed.

7.3.1 SynGNN with Cased BERT

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.66	0.83	0.74	598	CARDINAL	0.74	0.93	0.82	493
DATE	0.76	0.90	0.82	1041	DATE	0.70	0.90	0.79	584
EVENT	0.50	0.79	0.61	80	EVENT	0.29	0.74	0.42	23
FAC	0.35	0.62	0.45	151	FAC	0.45	0.68	0.54	66
GPE	0.90	0.93	0.92	1932	GPE	0.88	0.95	0.91	419
LANGUAGE	0.56	0.59	0.58	75	LANGUAGE	0.81	0.93	0.87	14
LAW	0.28	0.77	0.42	57	LAW	0.29	0.74	0.42	19
LOC	0.47	0.78	0.59	217	LOC	0.48	0.82	0.61	142
MONEY	0.35	0.69	0.46	61	MONEY	0.36	0.74	0.48	19
NORP	0.71	0.90	0.80	419	NORP	0.68	0.95	0.79	220
ORDINAL	0.86	0.96	0.91	170	ORDINAL	0.77	0.96	0.86	83
ORG	0.63	0.79	0.70	852	ORG	0.63	0.88	0.74	468
PERCENT	0.73	0.92	0.81	36	PERCENT	0.73	0.80	0.76	40
PERSON	0.83	0.92	0.88	1347	PERSON	0.82	0.95	0.88	808
PRODUCT	0.28	0.76	0.41	98	PRODUCT	0.33	0.66	0.44	67
QUANTITY	0.46	0.72	0.56	53	QUANTITY	0.41	0.81	0.55	101
TIME	0.47	0.78	0.59	214	TIME	0.43	0.79	0.56	71
W._OF_ART	0.33	0.59	0.42	130	W._OF_ART	0.27	0.67	0.39	90
Micro Avg	0.70	0.87	0.77	7531	Micro Avg	0.66	0.90	0.76	3727
Weigh. Avg	0.73	0.87	0.79	7531	Weigh. Avg	0.70	0.90	0.78	3727

Table 10: NER results for SynGNN gold with cased BERT for dependency and constituency dataset.

Table 10 shows the results on the test data for the SynGNN that was trained using gold syntax graphs and cased BERT model. It is performing markedly better than its BERT baseline counterpart for both dependency and constituency data. Comparing results of dependency and constituency data, the F1 results are similar, with 0.77 and 0.76. However, it is worth noting again, that the comparison of the performance with dependency and constituency graphs is not entirely fair at this point, because the dependency dataset contains both more sentences and more examples of Named Entities. Later on, this issue is revisited in the section about balanced data.

7.3.2 SynGNN with Uncased BERT

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.67	0.85	0.75	603	CARDINAL	0.69	0.92	0.79	493
DATE	0.76	0.92	0.83	1041	DATE	0.67	0.90	0.77	588
EVENT	0.29	0.71	0.41	80	EVENT	0.15	0.65	0.24	23
FAC	0.30	0.70	0.42	151	FAC	0.30	0.68	0.41	66
GPE	0.89	0.93	0.91	1930	GPE	0.80	0.93	0.86	419
LANGUAGE	0.48	0.62	0.54	77	LANGUAGE	0.52	0.93	0.67	14
LAW	0.12	0.63	0.20	57	LAW	0.17	0.74	0.27	19
LOC	0.36	0.70	0.48	217	LOC	0.40	0.73	0.52	142
MONEY	0.41	0.85	0.56	60	MONEY	0.23	0.63	0.34	19
NORP	0.67	0.89	0.77	423	NORP	0.65	0.88	0.75	221
ORDINAL	0.78	0.95	0.86	169	ORDINAL	0.63	0.98	0.77	83
ORG	0.48	0.72	0.58	846	ORG	0.45	0.86	0.59	468
PERCENT	0.62	0.92	0.74	36	PERCENT	0.58	0.85	0.69	40
PERSON	0.84	0.91	0.87	1357	PERSON	0.77	0.95	0.85	811
PRODUCT	0.17	0.63	0.27	95	PRODUCT	0.22	0.78	0.35	67
QUANTITY	0.46	0.77	0.57	53	QUANTITY	0.41	0.81	0.54	101
TIME	0.52	0.80	0.63	214	TIME	0.46	0.87	0.60	71
W._OF_ART	0.14	0.50	0.22	130	W._OF_ART	0.16	0.54	0.24	93
Micro Avg	0.63	0.86	0.72	7539	Micro Avg	0.56	0.88	0.69	3738
Weigh Avg	0.70	0.86	0.76	7539	Weigh. Avg	0.62	0.88	0.72	3738

Table 11: NER results for SynGNN gold with uncased BERT for dependency and constituency dataset.

Table 11 shows the results for the SynGNN using the uncased BERT model. In general, the SynGNN does again perform better than the baseline BERT model (cf. table 9). However, its performance is worse than that of the SynGNN trained with the cased data.

The low precision values despite generally high recall might seem counter-intuitive at first. They are most probably due to the fact that the catch-all NER tag O, denoting any word which is not a Named Entity, is not listed as a separate tag by the *seqeval* library. Thus, it does not influence the recall value, but is still influencing precision values, meaning that the lower values are due to non-NE words (tagged as O) being misclassified as a NE.

7.3.3 Label Weights Clipping

So far, only one model configuration was considered, with a fixed label weights clip value of 50. The clip value determines how much the weight of rare NE tags is boosted by clipping values which exceed the clip value.
Table 12 shows the SynGNN results when using a lower label clipping value of 20. Results for the dependency dataset stay the same, while the precision value with the constituency data is 2 points lower, going from 0.66 to 0.64.

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.65	0.86	0.74	598	CARDINAL	0.69	0.92	0.79	493
DATE	0.73	0.90	0.81	1041	DATE	0.67	0.90	0.77	584
EVENT	0.50	0.73	0.59	80	EVENT	0.27	0.70	0.39	23
FAC	0.37	0.67	0.48	151	FAC	0.47	0.67	0.55	66
GPE	0.91	0.93	0.92	1932	GPE	0.87	0.94	0.90	419
LANGUAGE	0.60	0.53	0.56	75	LANGUAGE	0.72	0.93	0.81	14
LAW	0.26	0.75	0.38	57	LAW	0.24	0.74	0.36	19
LOC	0.49	0.76	0.59	217	LOC	0.45	0.81	0.58	142
MONEY	0.32	0.66	0.43	61	MONEY	0.30	0.63	0.41	19
NORP	0.71	0.91	0.80	419	NORP	0.62	0.93	0.75	220
ORDINAL	0.84	0.95	0.89	170	ORDINAL	0.75	0.96	0.85	83
ORG	0.60	0.79	0.68	852	ORG	0.62	0.88	0.73	468
PERCENT	0.85	0.94	0.89	36	PERCENT	0.77	0.83	0.80	40
PERSON	0.83	0.93	0.88	1347	PERSON	0.84	0.94	0.89	808
PRODUCT	0.31	0.70	0.43	98	PRODUCT	0.36	0.78	0.49	67
QUANTITY	0.49	0.70	0.57	53	QUANTITY	0.39	0.79	0.52	101
TIME	0.48	0.78	0.59	214	TIME	0.43	0.85	0.57	71
W._OF_ART	0.34	0.68	0.45	130	W._OF_ART	0.26	0.71	0.38	90
Micro Avg	0.70	0.87	0.77	7531	Micro Avg	0.64	0.90	0.75	3727
Weigh. Avg	0.73	0.87	0.79	7531	Weigh. Avg	0.68	0.90	0.77	3727

Table 12: NER results for SynGNN gold with label weights clipping at 20 for dependency and constituency dataset.

It is also possible to set the clip value higher and thus incite the model to choose rare NE tags more often. Table 13 shows the SynGNN results when using a label clipping value

of 100. Here, results for the constituency dataset stay the same, while the precision value with the dependency data is 2 points lower, going from 0.70 to 0.68.
Overall, the clip value of 50 showed best results, which is why it was chosen as default clip value.

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.64	0.84	0.73	598	CARDINAL	0.73	0.92	0.82	493
DATE	0.74	0.90	0.81	1041	DATE	0.67	0.90	0.77	584
EVENT	0.47	0.79	0.59	80	EVENT	0.22	0.74	0.33	23
FAC	0.27	0.67	0.38	151	FAC	0.42	0.68	0.52	66
GPE	0.91	0.91	0.91	1932	GPE	0.87	0.95	0.91	419
LANGUAGE	0.59	0.67	0.63	75	LANGUAGE	0.65	0.93	0.76	14
LAW	0.26	0.77	0.39	57	LAW	0.28	0.68	0.39	19
LOC	0.45	0.77	0.57	217	LOC	0.48	0.80	0.60	142
MONEY	0.34	0.66	0.44	61	MONEY	0.33	0.68	0.45	19
NORP	0.74	0.90	0.81	419	NORP	0.65	0.94	0.77	220
ORDINAL	0.85	0.95	0.90	170	ORDINAL	0.74	0.96	0.84	83
ORG	0.59	0.78	0.67	852	ORG	0.62	0.88	0.73	468
PERCENT	0.77	0.92	0.84	36	PERCENT	0.77	0.83	0.80	40
PERSON	0.86	0.91	0.88	1347	PERSON	0.84	0.95	0.89	808
PRODUCT	0.23	0.69	0.35	98	PRODUCT	0.34	0.75	0.47	67
QUANTITY	0.43	0.77	0.55	53	QUANTITY	0.45	0.84	0.59	101
TIME	0.47	0.78	0.59	214	TIME	0.47	0.86	0.60	71
W._OF_ART	0.32	0.58	0.41	130	W._OF_ART	0.26	0.64	0.37	90
Micro Avg	0.68	0.86	0.76	7531	Micro Avg	0.65	0.90	0.76	3727
Weigh. Avg	0.73	0.86	0.78	7531	Weigh. Avg	0.69	0.90	0.77	3727

Table 13: NER results for SynGNN gold with label weights clipping at 100 for dependency and constituency dataset.

7.4 SynGNN with Generated Syntax Graphs

So far, only gold-standard, hand-annotated syntax graphs were used to conduct the experiments. Obviously, these are costly to create and only available for a choice number of datasets, which limits the use of the SynGNN architecture. For real-world applications it would be preferable to be able to automatically create those syntax graphs on-the-fly when analysing any kind of text.
To explore this possibility, the constituency syntax graphs for the ICE-GB dataset were also created automatically with spaCy (Honnibal and Montani (2022)) and the Berkeley Neural Parser (Kitaev et al. (2019)). In this way, it can be evaluated if generated syntax graphs can be used instead of the gold-standard ones and how much this influences

the model performance. The experiments were again conducted with the cased and uncased BERT model, using the respective tokenizer for each.

7.4.1 SynGNN with Cased BERT

Table 14 shows the SynGNN results using the generated syntax graphs for the constituency data and the cased BERT model. The model configuration is based on the best configuration found during the previous experiments, which was discussed in 7.3.1 *SynGNN with Cased BERT*. The F1 score of 0.76 is the same as with the gold-standard constituency graphs.

	Constituency			
NE type	Prec.	Recall	F1	#
CARDINAL	0.75	0.91	0.82	493
DATE	0.71	0.90	0.79	584
EVENT	0.27	0.83	0.40	23
FAC	0.37	0.68	0.48	66
GPE	0.89	0.94	0.91	421
LANGUAGE	0.72	0.93	0.81	14
LAW	0.27	0.74	0.39	19
LOC	0.47	0.79	0.59	142
MONEY	0.31	0.63	0.41	19
NORP	0.73	0.93	0.82	220
ORDINAL	0.76	0.96	0.85	84
ORG	0.57	0.89	0.70	471
PERCENT	0.67	0.78	0.72	40
PERSON	0.83	0.95	0.88	808
PRODUCT	0.37	0.73	0.49	67
QUANTITY	0.41	0.82	0.55	101
TIME	0.45	0.80	0.57	71
W._OF_ART	0.31	0.69	0.42	90
Micro Avg	0.66	0.89	0.76	3733
Weigh. Avg	0.69	0.89	0.78	3733

Table 14: NER results for SynGNN with generated constituency graphs and cased BERT.

7.4.2 SynGNN with Uncased BERT

Table 15 shows the SynGNN results using the generated syntax graphs for the constituency data and the uncased BERT model. The model configuration is based on the best configuration found during the previous experiments, which was discussed in 7.3.2 *SynGNN with Unased BERT* As before, the F1 score of 0.69 is the same as when using the gold-standard constituency graphs.

	Constituency			
NE type	Prec.	Recall	F1	#
CARDINAL	0.71	0.92	0.81	493
DATE	0.71	0.91	0.80	588
EVENT	0.13	0.70	0.21	23
FAC	0.31	0.65	0.42	66
GPE	0.80	0.94	0.86	421
LANGUAGE	0.61	1.00	0.76	14
LAW	0.16	0.63	0.25	19
LOC	0.37	0.75	0.50	142
MONEY	0.19	0.68	0.29	19
NORP	0.64	0.90	0.75	221
ORDINAL	0.68	0.98	0.80	84
ORG	0.46	0.86	0.60	471
PERCENT	0.73	0.80	0.76	40
PERSON	0.81	0.95	0.87	811
PRODUCT	0.19	0.72	0.30	67
QUANTITY	0.34	0.80	0.48	101
TIME	0.47	0.87	0.61	71
W._OF_ART	0.18	0.52	0.27	93
Micro Avg	0.57	0.89	0.69	3744
Weigh. Avg	0.64	0.89	0.73	3744

Table 15: NER results for SynGNN with generated constituency graphs and uncased BERT.

7.5 SynGNN with Balanced Data

As was seen in the earlier experiments, the amount of data in the dependency and constituency dataset is not the same. While it is certainly useful to include all available data when evaluating the general performance of the SynGNN model against the BERT baseline, it makes a direct comparison between the SynGNNs trained on these datasets a bit unfair.

To remedy this, the dependency dataset, which contains roughly 40k sentences as compared to 30k in the constituency dataset and also more NE tags, was trimmed down to match the size of the constituency dataset. This is done by randomly discarding a set fraction of sentences that do not contain NEs as well as a set fraction of sentences containing NEs. The fractions are tweaked until both the amount of sentences and the amount of NEs in the datasets matches.

7.5.1 SynGNN Cased

	Dependency Bal.					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.60	0.85	0.70	301	CARDINAL	0.74	0.93	0.82	493
DATE	0.63	0.86	0.73	536	DATE	0.70	0.90	0.79	584
EVENT	0.36	0.72	0.48	50	EVENT	0.29	0.74	0.42	23
FAC	0.26	0.65	0.37	69	FAC	0.45	0.68	0.54	66
GPE	0.88	0.91	0.89	981	GPE	0.88	0.95	0.91	419
LANGUAGE	0.53	0.60	0.56	47	LANGUAGE	0.81	0.93	0.87	14
LAW	0.28	0.77	0.41	31	LAW	0.29	0.74	0.42	19
LOC	0.41	0.68	0.51	98	LOC	0.48	0.82	0.61	142
MONEY	0.37	0.82	0.51	28	MONEY	0.36	0.74	0.48	19
NORP	0.58	0.89	0.70	201	NORP	0.68	0.95	0.79	220
ORDINAL	0.76	0.94	0.84	79	ORDINAL	0.77	0.96	0.86	83
ORG	0.51	0.80	0.62	412	ORG	0.63	0.88	0.74	468
PERCENT	0.59	0.87	0.70	15	PERCENT	0.73	0.80	0.76	40
PERSON	0.77	0.90	0.83	640	PERSON	0.82	0.95	0.88	808
PRODUCT	0.20	0.57	0.29	44	PRODUCT	0.33	0.66	0.44	67
QUANTITY	0.28	0.61	0.39	31	QUANTITY	0.41	0.81	0.55	101
TIME	0.42	0.79	0.55	112	TIME	0.43	0.79	0.56	71
W._OF_ART	0.16	0.58	0.24	59	W._OF_ART	0.27	0.67	0.39	90
Micro Avg	0.60	0.85	0.70	3734	Micro Avg	0.66	0.90	0.76	3727
Weigh. Avg	0.66	0.85	0.73	3734	Weigh. Avg	0.70	0.90	0.78	3727

Table 16: NER results for SynGNN gold with cased BERT for balanced dependency and constituency dataset with 30k sentences each

To create a balanced dataset from the dependency dataset matching the amount of data

in the constituency dataset, 4 percent of all sentences not containing NEs is deleted, as well as 49.5 percent of all sentences containing NEs. This results in a total number of 29428 sentences and 25095 NE tags. The test data now contains 3734 NE tags, matching the 3727 in the constituency test dataset.
With the amount of data balanced like this, the results are very interesting (see table 16). Now, the SynGNN trained on dependency graphs has a clear disadvantage over the one trained with constituency graphs. The F1 score is 0.7 for the dependency graphs and 0.76 for the constituency graphs. Also, now there is only one NE tag, MONEY, where the dependency SynGNN performs better than the constituency one, and even that is only by 3 points.

7.5.2 SynGNN Uncased

	Dependency Bal.					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.56	0.87	0.68	290	CARDINAL	0.69	0.92	0.79	493
DATE	0.67	0.88	0.76	521	DATE	0.67	0.90	0.77	588
EVENT	0.24	0.64	0.35	44	EVENT	0.15	0.65	0.24	23
FAC	0.23	0.60	0.34	77	FAC	0.30	0.68	0.41	66
GPE	0.87	0.91	0.89	958	GPE	0.80	0.93	0.86	419
LANGUAGE	0.47	0.63	0.53	32	LANGUAGE	0.52	0.93	0.67	14
LAW	0.13	0.60	0.21	35	LAW	0.17	0.74	0.27	19
LOC	0.30	0.74	0.43	109	LOC	0.40	0.73	0.52	142
MONEY	0.40	0.88	0.55	26	MONEY	0.23	0.63	0.34	19
NORP	0.55	0.88	0.68	181	NORP	0.65	0.88	0.75	221
ORDINAL	0.68	0.95	0.79	77	ORDINAL	0.63	0.98	0.77	83
ORG	0.45	0.70	0.55	430	ORG	0.45	0.86	0.59	468
PERCENT	0.83	0.95	0.88	20	PERCENT	0.58	0.85	0.69	40
PERSON	0.76	0.90	0.82	704	PERSON	0.77	0.95	0.85	811
PRODUCT	0.09	0.50	0.15	36	PRODUCT	0.22	0.78	0.35	67
QUANTITY	0.45	0.84	0.58	31	QUANTITY	0.41	0.81	0.54	101
TIME	0.49	0.86	0.63	111	TIME	0.46	0.87	0.60	71
W._OF_ART	0.17	0.40	0.24	67	W._OF_ART	0.16	0.54	0.24	93
Micro Avg	0.57	0.84	0.68	3749	Micro Avg	0.56	0.88	0.69	3738
Weigh. Avg	0.64	0.84	0.72	3749	Weigh. Avg	0.62	0.88	0.72	3738

Table 17: NER results for SynGNN gold with uncased BERT for balanced dependency and constituency dataset with 30k sentences each

The same experiment was also repeated using the uncased BERT model. Since the uncased dependency dataset contains a slightly different amount of sentences overall because of the mapping between syntax graphs and sentences, the deletion rates had to be re-calculated to fit them. Here, 2% of all sentences without NEs are discarded

and 48.8% of all sentences containing NEs. This results in a total of 29962 sentences and 25456 NE tags, with 3749 of those in the test set, matching the number of NE tags in the constituency test set as close as possible.
As expected, the results are not as good as with the cased dataset (see table 17). Also, the difference between using dependency and constituency graphs for the SynGNN training is much less pronounced, almost nonexistent. The F1 score for the dependency-trained SynGNN is 0.68 and with the constituency-trained SynGNN only 1 point more, 0.69.

7.6 Enhanced BERT Model

The baseline BERT model seems to be performing not as well as the SynGNN. However, the question still arises, how much of this is due to the inclusion of the information of the syntax graphs. It is also possible, for example, that the SynGNN model in reality profits more from e.g. the inclusion of the label weights or having 15 additional epochs of training. These experiments are designed to clarify these questions.

7.6.1 BERT Baseline with Label Weights

Table 18 shows the NER results for the pre-trained cased BERT with label weights activated for the cross-entropy loss calculation as in the SynGNN. One can see that expectedly, the recall stays the same at 0.73 with the dependency dataset. It rises to 0.71 with the constituency dataset, from 0.65 with the original baseline model. Precision, however, suffers immensely with both models and drops to 0.20 and 0.16 respectively. The original baseline model achieves much higher values.
So, this points to the conclusion that the improvements seen with the SynGNN are not only due to the inclusion of label weights.

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.18	0.77	0.30	611	CARDINAL	0.22	0.81	0.34	494
DATE	0.21	0.65	0.31	1045	DATE	0.16	0.64	0.26	591
EVENT	0.13	0.55	0.20	80	EVENT	0.07	0.48	0.12	23
FAC	0.11	0.44	0.17	151	FAC	0.07	0.29	0.11	66
GPE	0.40	0.82	0.54	1936	GPE	0.24	0.83	0.37	423
LANGUAGE	0.13	0.62	0.21	77	LANGUAGE	0.11	0.93	0.19	14
LAW	0.11	0.61	0.19	57	LAW	0.07	0.47	0.11	19
LOC	0.15	0.64	0.25	217	LOC	0.12	0.62	0.20	142
MONEY	0.18	0.69	0.28	61	MONEY	0.07	0.63	0.13	19
NORP	0.18	0.86	0.29	422	NORP	0.17	0.83	0.28	221
ORDINAL	0.26	0.91	0.40	171	ORDINAL	0.18	0.93	0.30	84
ORG	0.17	0.63	0.27	857	ORG	0.16	0.66	0.25	471
PERCENT	0.06	0.42	0.10	36	PERCENT	0.08	0.40	0.13	40
PERSON	0.29	0.78	0.43	1370	PERSON	0.25	0.79	0.38	814
PRODUCT	0.10	0.56	0.17	98	PRODUCT	0.08	0.63	0.14	67
QUANTITY	0.07	0.36	0.12	53	QUANTITY	0.07	0.28	0.11	101
TIME	0.18	0.66	0.29	214	TIME	0.12	0.66	0.20	71
W._OF_ART	0.10	0.43	0.16	130	W._OF_ART	0.06	0.40	0.10	93
Micro Avg	0.20	0.73	0.32	7586	Micro Avg	0.16	0.71	0.26	3753
Weigh. Avg	0.25	0.73	0.37	7586	Weigh. Avg	0.18	0.71	0.29	3753

Table 18: NER results for BERT baseline model cased trained with label weights activated

7.6.2 BERT Baseline with Additional Training

The second experiment consists of taking the pre-trained Bert baseline model and training it further with an additional 12 epochs of training to simulate SynGNN training. The training is done with the same configuration as the SynGNN models, i.e. also using label weights and the same parameters. The only difference is, that the SynGNN is not active.

This experiment is designed to investigate the possibility that the BERT baseline may have learnt the same things, even without access to the syntax graphs in those additional training epochs for the SynGNN model.

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.32	0.80	0.46	611	CARDINAL	0.38	0.87	0.53	506
DATE	0.44	0.79	0.56	1045	DATE	0.34	0.81	0.48	593
EVENT	0.26	0.60	0.36	80	EVENT	0.20	0.73	0.31	22
FAC	0.20	0.52	0.29	151	FAC	0.19	0.61	0.29	67
GPE	0.69	0.88	0.77	1936	GPE	0.43	0.91	0.59	424
LANGUAGE	0.29	0.60	0.39	77	LANGUAGE	0.30	0.87	0.45	15
LAW	0.21	0.70	0.32	57	LAW	0.16	0.68	0.26	19
LOC	0.22	0.71	0.34	217	LOC	0.28	0.72	0.40	143
MONEY	0.24	0.75	0.37	61	MONEY	0.16	0.67	0.26	18
NORP	0.36	0.88	0.51	422	NORP	0.36	0.95	0.52	221
ORDINAL	0.50	0.95	0.66	171	ORDINAL	0.35	0.96	0.51	84
ORG	0.34	0.70	0.46	857	ORG	0.33	0.80	0.47	476
PERCENT	0.16	0.61	0.25	36	PERCENT	0.14	0.46	0.22	41
PERSON	0.56	0.85	0.68	1370	PERSON	0.51	0.90	0.65	820
PRODUCT	0.22	0.66	0.33	98	PRODUCT	0.25	0.72	0.37	69
QUANTITY	0.16	0.55	0.25	53	QUANTITY	0.23	0.66	0.34	100
TIME	0.32	0.75	0.45	214	TIME	0.25	0.78	0.38	73
W._OF_ART	0.16	0.50	0.24	130	W._OF_ART	0.11	0.45	0.18	101
Micro Avg	0.40	0.80	0.53	7586	Micro Avg	0.33	0.83	0.47	3792
Weigh. Avg	0.47	0.80	0.58	7586	Weigh. Avg	0.37	0.83	0.51	3792

Table 19: NER results for BERT baseline model cased with additional training epochs

8 Discussion of Results

In this section, the experiment results are discussed in more detail and possible explanations for some of the observations are given. In particular, four questions are to be answered:

- Does the SynGNN improve over the baseline BERT model?
- How do gold-standard and generated syntax graphs compare?
- How do the SynGNNs using dependency and constituency graphs compare?
- How do the results obtained compare to those observed by Sachan et al. (2021)?

Table 20 shows on overview of results from the BERT baseline model and the best-performing SynGNN models grouped by cased and uncased BERT model. These are the results obtained when using the full datasets available. Table 21 shows on overview of results from the BERT baseline model and SynGNN models when using matching dataset sizes for the constituency and dependency dataset.

	Dependency			Constituency		
Model	Prec.	Recall	F1	Prec.	Recall	F1
BERT Cased	0.68	0.73	0.70	0.62	0.65	0.63
SynGNN Gold Cased	0.70	0.87	0.77	0.66	0.90	0.76
SynGNN Gen. Cased	-	-	-	0.66	0.89	0.76
BERT Uncased	0.67	0.71	0.69	0.60	0.59	0.60
SynGNN Gold Uncased	0.63	0.86	0.72	0.56	0.88	0.69
SynGNN Gen. Uncased	-	-	-	0.57	0.89	0.69

Table 20: Overview of NER results for BERT baseline and SynGNN on full datasets

	Dependency			Constituency		
Model	Prec.	Recall	F1	Prec.	Recall	F1
BERT Cased Bal.	0.58	0.63	0.6	0.62	0.65	0.63
SynGNN Gold Cased Bal.	0.60	0.85	0.70	0.66	0.90	0.76
BERT Uncased Bal.	0.62	0.67	0.64	0.60	0.59	0.60
SynGNN Gold Uncased Bal.	0.57	0.84	0.68	0.56	0.88	0.69

Table 21: Overview of NER results for BERT baseline and SynGNN on balanced datasets

8.1 Comparison of BERT baseline and SynGNN

To compare the performance of the baseline BERT model and the SynGNN, the results given in table 20 when using the full datasets are evaluated. Here, the focus is not on inter-dataset comparisons but rather intra-dataset: comparing each SynGNN with its own baseline model.
The first grouping in the table shows the cased BERT baseline and SynGNN models. With the cased BERT encoder, the SynGNN models with gold dependency or constituency graphs both improve over their respective baseline models with a baseline F1 of 0.70 (dependency) and 0.63 (constituency). The SynGNN reaches an F1 score of 0.77 using the dependency graphs and 0.76 with the constituency graphs. Remarkably, while the performance of the two SynGNNs is similar, the one using constituency graphs has improved 13 points over its baseline F1, while the SynGNN using the dependency graphs improves only 7 points over the baseline BERT model. This higher F1 score over the baseline model with the constituency graphs is mostly due to a large jump in recall. The baseline model correctly predicts 65% of all members of a given class, while the SynGNN manages to correctly predict 90% of all members of a class as belonging to this class. The difference in precision between the pre-trained BERT and SynGNN is not as great, an improvement of 2 points, 0.68 to 0.70, with the dependency graphs and an improvement of 4 points, 0.62 to 0.66, using the constituency graphs.
The second grouping in table 20 compares model performance when using the uncased BERT model. The BERT baseline model performs markedly worse when using the uncased texts of the constituency dataset. Recall drops from 0.65 to 0.59 and overall F1 also drops 3 points to 0.60. The difference is less noticeable with the dependency dataset: here, the baseline model only drops 2 points in recall and 1 in the F1 score.

8.2 Comparison of Gold-standard and Generated Syntax Graphs

The basis for comparing the model performance with hand-annotated gold syntax trees to the generated trees by a state-of-the-art constituency parser is again table 20. Using the cased BERT model, the comparison reveals that there are only minor differences in performance regarding the SynGNN. The F1 score for both the model trained with gold-standard graphs and the one using the generated graphs is the same, 0.76. The only difference is a slightly better recall of 0.90 with the gold trees compared to 0.89 with the generated trees. Comparing the performance of hand-annotated and generated constituency trees with the uncased SynGNN, there again is no marked difference, with

the F1 scores being the same.
These results indicate that it is possible to replace the hand-annotated trees in the SynGNN with more readily available generated ones without substantially sacrificing model performance in the process.

8.3 Comparison of SynGNN with Dependency and Constituency Graphs

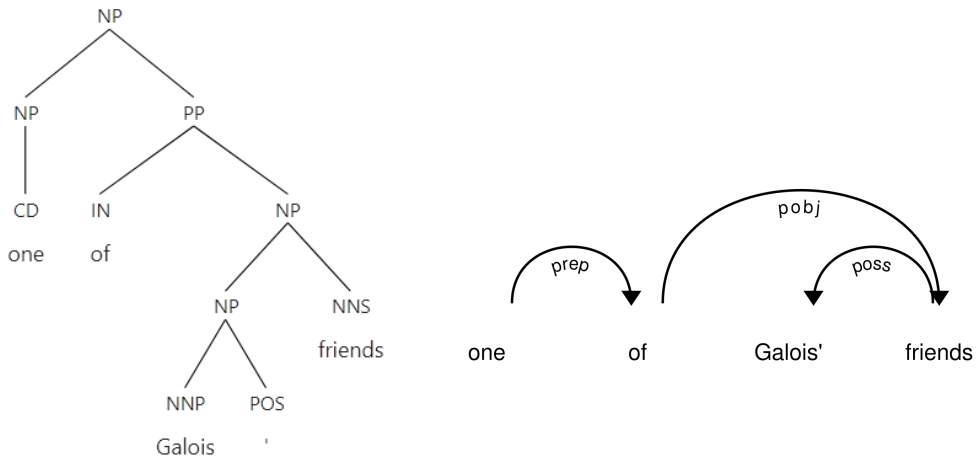
The main comparison of the SynGNN model performance using dependency and constituency graphs is done with the results obtained with the balanced datasets using the gold-standard syntax graphs, given in 21.
Evaluating the performance with the cased BERT model, the SynGNN model trained with constituency graphs performs better than the one trained on dependency graphs. Both precision and recall are better, resulting in an F1 improvement from 0.7 to 0.76. Also, the constituency SynGNN is able to improve more over its BERT baseline model than the dependency SynGNN does. The dependency baseline is 0.6 and improves to 0.7, the constituency baseline is 0.63 with the constituency SynGNN reaching 0.76.
Looking at the uncased SynGNNs, performances are similar to those seen with the full, unbalanced datasets. Again, there is no clear difference between using dependency graphs and constituency graphs - the F1 scores are almost the same for both, differing in 1 point (0.68 to 0.69).

Moving on from the general model performance, there are also some interesting phenomena to be observed regarding specific NE tags. One thing to note is that, with the balanced data, the constituency-trained SynGNN delivers consistently higher scores for all NE tags except MONEY.
Table 22 shows a selection of NE tags where the metrics differ most noticeably between the SynGNNs trained with dependency and constituency trees. This was arbitrarily defined as being the case when there are more than 10 points difference in the F1 values. Also, only those results were included, where the highest F1 reached is greater than 0.5, i.e. higher than chance. Using these criteria, the metrics for CARDINAL, FAC, LANGUAGE, ORG, and QUANTITY are better when using constituency trees instead of dependency trees.
A comparison of example sentences parsed with both dependency and constituency grammar might shed some light on possible explanations why some of these NE tags are classified better when using one or the other style. All of the NE examples are

	Dependency					Constituency			
NE type	Prec.	Recall	F1	#	NE type	Prec.	Recall	F1	#
CARDINAL	0.60	0.85	0.70	301	CARDINAL	0.74	0.93	0.82	493
FAC	0.26	0.65	0.37	69	FAC	0.45	0.68	0.54	66
LANGUAGE	0.53	0.60	0.56	47	LANGUAGE	0.81	0.93	0.87	14
ORG	0.51	0.80	0.62	412	ORG	0.63	0.88	0.74	468
QUANTITY	0.28	0.61	0.39	31	QUANTITY	0.41	0.81	0.55	101

Table 22: Selected NER results for SynGNN gold with cased BERT for dependency and constituency dataset.

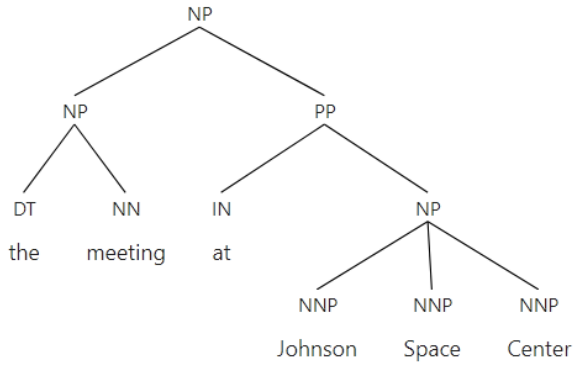
real examples found in the Universal Dependencies dataset. The visualization of the constituency parses is done with Benepar (Stern (2022)) and the visualizations of the dependency parses is generated by the spaCy visualization tool displaCy (Honnibal and Montani (2022)).



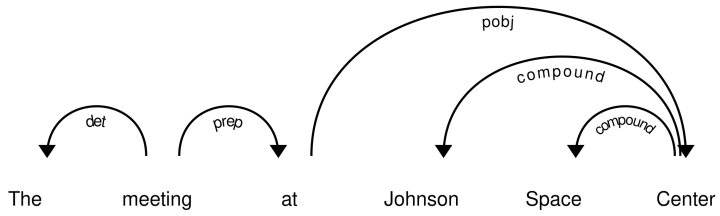
(a) Constituency tree (b) Dependency tree

Figure 11: Syntax graphs for NE tag *CARDINAL*

For the NE type *CARDINAL*, the SynGNN using the constituency graphs shows both higher precision (0.6 to 0.74) and recall (0.85 to 0.93). Looking at the syntax graphs in figure 11 for the phrase *one of Galois' friends*, the cardinal value *one* is modelled differently in the two grammar types. In the constituency tree, it is explicitly marked as a cardinal value (*CD*). In the dependency tree it is modelled as the head of a preposition with no indication of it being a cardinal. So, it is perhaps not surprising that the constituency SynGNN is better at detecting cardinal values.



(a) Constituency tree

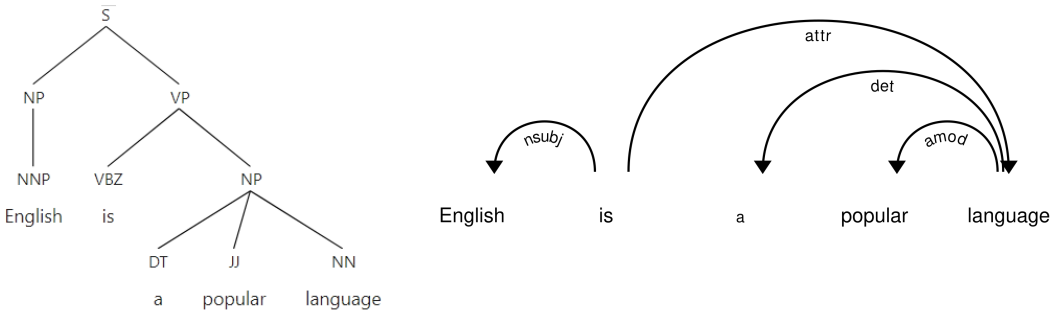


(b) Dependency tree

Figure 12: Syntax graphs for NE tag *FAC*

The NE tag FAC stands for names of buildings, such as *Johnson Space Center*. Here, recall is almost the same with both types of syntax graphs, but precision rises from 0.26 to 0.45, with an almost identical total number of entities tagged as FAC (about 66). So, the dependency SynGNN correctly identifies the same amount of FAC entities, but must falsely classify other NEs as belonging to the group of FACs. Figure 12 shows syntax graphs containing this NE. In constituency grammar, it is modelled as a noun phrase (NP) node with three children all classified as proper nouns (NNP). The dependency tree models it as *Center* being the head of two compounds, *Johnson* and *Space*. This is very specific and it is understandable that the GAT layer will pick up on the connection in both cases. For the constituency tree, the information of all three children gets aggregated by default. The dependency tree also aggregates information from all three nodes, because the head node *Center* is treated as a neighbour of itself by introducing the self-loops.

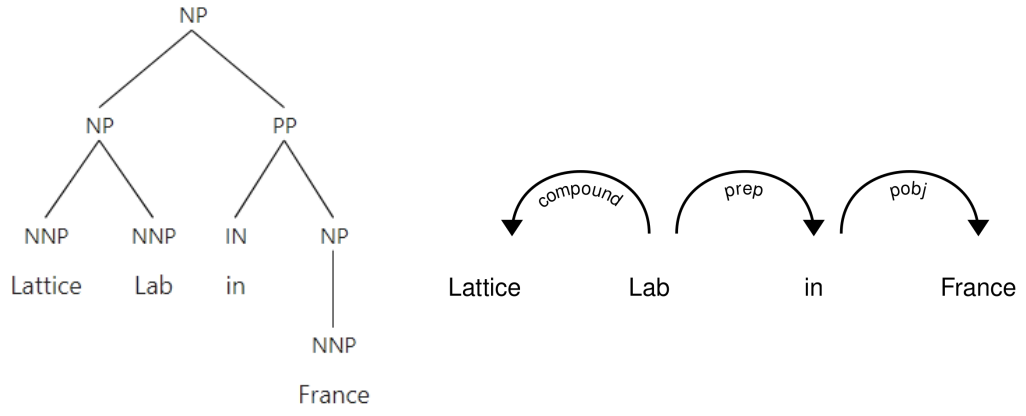
This might explain the high success rate concerning recall in both cases, but still with the dependency graphs there must be a similar construction, which is confused with the one characterizing FAC values, to cause the drop in precision.



(a) Constituency tree (b) Dependency tree

Figure 13: Syntax graphs for NE tag *LANGUAGE*

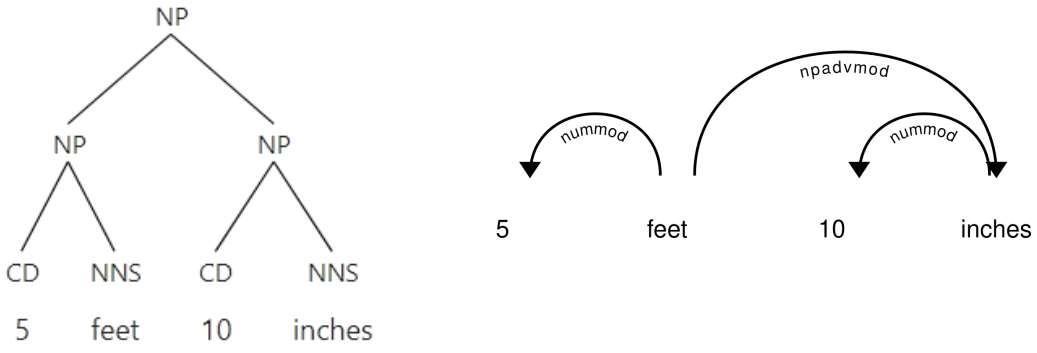
The difference when classifying LANGUAGE-type NEs is perhaps the most surprising. Both precision and recall are improved, bringing the F1 score from 0.56 to 0.87. Looking at the syntax graphs in figure 13, the language name *English* is modelled as a proper noun in the constituency tree. The dependency tree focuses on its syntactic function only, classifying it as *nsubj*, a subject noun to the verb *is*. With examples like these, it is likely that the classification by syntactic function is simply too generic and not specific enough to distinguish a LANGUAGE NE from any other subject noun when using dependency grammar.
Also, the constituency dataset has markedly less LANGUAGE entities to learn, only 14 whereas there are 47 samples in the dependency dataset.



(a) Constituency tree (b) Dependency tree

Figure 14: Syntax graphs for NE tag *ORG*

Compared to this, the difference in F1 for the NE tag ORG, denoting an organization name, is much more modest, it increases from 0.62 to 0.74. Recall is already high with the dependency trees, 0.8, and increases to 0.88. Precision also increases from about half to 0.63. Looking at the syntax graphs in figure 14, the modelling for the constituency graph is similar to that of LANGUAGE. The organization name *Lattice Lab* is parsed as a noun phrase consisting of two proper nouns.
In the dependency parse, we see that *Lab* is the head of a compound dependency with *Lattice* as dependant. This compound dependency is specifically reserved for compound words, so it does give a solid hint that these two words may form a NE of some kind. This might explain why recall is quite good with both syntactic approaches this time. Language names typically only consist of one word and not compounds, so the dependency SynGNN has no additional information to go on with those, resulting in worse recall.



(a) Constituency tree (b) Dependency tree

Figure 15: Syntax graphs for NE tag *QUANTITY*

Figure 15 shows the syntax graphs for the QUANTITY phrase *5 feet 10 inches*. For QUANTITY NE tags, classification success is still quite low regardless of what syntax graph is used: the dependency-based SynGNN reaches an F1 score of 0.39 while the constituency-based SynGNN barely arrives at an F1 of 0.55. As can be seen from the example, QUANTITY phrases can get quite complex. Constituency grammar models the phrase as being two noun phrases (NP), each consisting of a cardinal value and a plural noun. Both NPs are combined in one NP node to form the quantity expression.
In the dependency tree, *feet* is the head, with two children: *5* as numerical modifier (=nummod) and the noun phrase used as adverbial modifier (=npadvmod) *10 inches* where *inches* is head of its numerical modifer *10*. Thus, while both grammar types state

that there are numbers involved, the information given in the constituency tree is still more specific. It states that a single QUANTITY phrase is made up of a cardinal number and a plural noun in this case. The dependency tree only gives us the grammatical function of numerical modifier without distinguishing between cardinal or ordinal.
Still, QUANTITY phrases seem to be complex enough that both SynGNN types do not manage to predict them with high certainty.

8.4 Comparison of Results with Sachan et al.

Comparing these SynGNN results with those obtained by Sachan et al. (2021), who also tested their model with a NER task, there are striking differences. They use the OntoNotes 5.0 dataset (Linguistic Data Consortium (2013)), which is also tagged with 18 different NER tags. Their baseline BERT model reaches an F1 of 89.18 while the Late Fusion model with dependency graphs reaches only 88.97, so there is no improvement over the baseline. These results can be due to a variety of factors. The dataset is roughly double the size of the Universal Dependency dataset used here (714k words versus 1,325k words). So maybe the baseline BERT model already has enough data to perform well with the task on its own. Also, the domains from which the texts are taken are very homogenous, almost two thirds of the data come from news or broadcast news. This might also help the BERT model with its performance, so that the additional syntax data cannot be leveraged for better results. Looking towards differences in model construction, the SynGNN used in this work differs in some ways from the one by Sachan et al. (2021). Most importantly, the edge features are taken into account, which might explain better performance with the dependency graphs. The constituency graphs however, do not have any edge features and still perform better than the baseline model.

9 Conclusion and Future Work

With sufficiently large datasets, such as the one used by Sachan et al. (2021), BERT itself seems to be able to learn the relevant syntactic interdependencies on its own. This is also shown in the studies mentioned in the introduction. So, it is suspected that a larger dataset improves the BERT baseline results enough so that adding the SynGNN does not result in relevant performance gains. However, for the smaller datasets used in this study, adding the SynGNN did make a marked difference in performance, especially improving recall scores.

Comparing the performance of the SynGNN with dependency and constituency syntax graphs, the SynGNN trained on dependency graphs performed worse than the one trained on constituency graphs. The recommendation for NER-style tasks would therefore be to use constituency graphs over dependency graphs, when available.

A third important finding suggests that the constituency graphs generated by a state-of-the-art parser are good enough to help the SynGNN model improve its guesses as well as the hand-annotated ones. So, in brief, the evidence seems to suggest that it is advantageous to use a SynGNN over BERT when the amount of data is small. Since generated graphs performed comparably to the hand-annotated ones, it would be relatively easy to build a model using those automatically generated constituency or dependency graphs for any input text.

This also ties in with the outlook on possible future routes of research and development. It might be worthwhile to build such a SynGNN model, which directly generates the syntax graphs for any input text. So far, they are generated and stored beforehand. To investigate the SynGNNs capabilities, as well as differences between dependency and constituency graphs, further, other tasks than the NER task highlighted here should be considered. Possible options include Semantic Role Labeling and Relation Extraction. The results by Sachan et al. (2021) suggest that with Relation Extraction tasks, dependency graphs may have an advantage over constituency-style graphs because of their capturing of syntactic relations between words. Using these tasks would also offer the chance to directly compare results with those by Sachan et al. (2021), by comparing results on the same tasks and datasets.

References

CrossEntropyLoss — PyTorch 1.12 documentation, October 2022. URL https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Online; accessed 16. Oct. 2022].

A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, 2019.

J. Bai, Y. Wang, Y. Chen, Y. Yang, J. Bai, J. Yu, and Y. Tong. Syntax-bert: Improving pre-trained transformers with syntax trees. *CoRR*, abs/2103.04350, 2021. URL https://arxiv.org/abs/2103.04350.

S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *CoRR*, abs/2105.14491, 2021. URL https://arxiv.org/abs/2105.14491.

S. Buchholz and E. Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June 2006. Association for Computational Linguistics. URL https://aclanthology.org/W06-2920.

N. Chomsky. *The Logical Structure of Linguistic Theory*. Springer US, 1956/1975. ISBN 9780306307607.

M.-C. de Marneffe and J. Nivre. Dependency grammar. *Annual Review of Linguistics*, 5(1):197–218, 2019. doi: 10.1146/annurev-linguistics-011718-011842. URL https://doi.org/10.1146/annurev-linguistics-011718-011842.

M.-C. de Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062_Paper.pdf.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

V.A. Fromkin, B. Hayes, S. Curtiss, A. Szabolcsi, T. Stowell, E. Stabler, D. Sportiche, H. Koopman, P. Keating, P. Munro, et al. *Linguistics: An Introduction to Linguistic Theory*, chapter Syntax I: Argument Structure and Phrase Structure. Wiley, 2013. ISBN 9781118670910.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

D. Goel and R. Sharma. Leveraging dependency grammar for fine-grained offensive language detection using graph convolutional networks, 2022. URL https://arxiv.org/abs/2205.13164.

Y. Goldberg. Assessing bert's syntactic abilities. *CoRR*, abs/1901.05287, 2019. URL http://arxiv.org/abs/1901.05287.

D. Hendrycks and K. Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. URL http://arxiv.org/abs/1606.08415.

M. Honnibal and I. Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

M. Honnibal and I. Montani. spaCy · Industrial-strength Natural Language Processing in Python, October 2022. URL https://spacy.io. [Online; accessed 4. Oct. 2022].

Hugging Face. bert-base-cased, October 2022a. URL https://huggingface.co/bert-base-cased. [Online; accessed 12. Oct. 2022].

Hugging Face. bert-base-uncased, October 2022b. URL https://huggingface.co/bert-base-uncased. [Online; accessed 12. Oct. 2022].

D. Jurafsky and J. H. Martin. Speech and language processing (3rd edition). unpublished draft at https://web.stanford.edu/~jurafsky/slp3/, 2021.

N. Kitaev and D. Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1249. URL https://www.aclweb.org/anthology/P18-1249.

N. Kitaev, S. Cao, and D. Klein. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1340. URL https://www.aclweb.org/anthology/P19-1340.

Linguistic Data Consortium. OntoNotes Release 5.0, October 2013. URL https://catalog.ldc.upenn.edu/LDC2013T19. [Online; accessed 22. Oct. 2022].

Linguistic Data Consortium. English News Text Treebank: Penn Treebank Revised, October 2022. URL https://catalog.ldc.upenn.edu/LDC2015T13. [Online; accessed 8. Oct. 2022].

Z. Luo. Have attention heads in BERT learned constituency grammar? *CoRR*, abs/2102.07926, 2021. URL https://arxiv.org/abs/2102.07926.

M.-C Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, Joakim Nivre, and C.D. Manning. Universal stanford dependencies: A cross-linguistic typology. *Proceedings of the 9Th International Conference on Language Resources and Evaluation (LREC)*, pages 4585–4592, 01 2014.

H. Nakayama. seqeval: A python framework for sequence labeling evaluation, 2018. URL https://github.com/chakki-works/seqeval. Software available from https://github.com/chakki-works/seqeval.

G. Nelson, S. Wallis, and B. Aarts. *Exploring Natural Language: Working with the British Component of the International Corpus of English*. John Benjamins, 2002. URL https://www.jbe-platform.com/content/books/9789027275356.

J. Nivre, M.-C. de Marneffe, F. Ginter, J. Hajic, C. D. Manning, S. Pyysalo, S. Schuster, F. M. Tyers, and D. Zeman. Universal dependencies v2: An evergrowing multilingual treebank collection. *CoRR*, abs/2004.10643, 2020. URL https://arxiv.org/abs/2004.10643.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Education, 2016. ISBN 9781292153971.

D. Sachan, Y. Zhang, P. Qi, and W. L. Hamilton. Do syntax trees help pre-trained transformers extract information? In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2647–2661, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.228. URL https://aclanthology.org/2021.eacl-main.228.

scikit-learn. sklearn.utils.class_weight.compute_class_weight, October 2022. URL https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html. [Online; accessed 16. Oct. 2022].

R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387.

M. Stern. Berkeley Neural Parser Web Implementation, August 2022. URL https://parser.kitaev.io. [Online; accessed 4. Oct. 2022].

E. Strubell, P. Verga, D. Andor, D. Weiss, and A. McCallum. Linguistically-informed self-attention for semantic role labeling. *CoRR*, abs/1804.08199, 2018. URL http://arxiv.org/abs/1804.08199.

D. Sundararaman, V. Subramanian, G. Wang, S. Si, D. Shen, D. Wang, and L. Carin. Syntax-infused transformer and BERT models for machine translation and natural language understanding. *CoRR*, abs/1911.06156, 2019. URL http://arxiv.org/abs/1911.06156.

L. Tesnière. *Élements de syntaxe structurale*. C. Klincksieck, 1959.

UCL Survey of English Usage. ICE-GB, May 2020. URL https://www.ucl.ac.uk/english-usage/projects/ice-gb/index.htm. [Online; accessed 8. Oct. 2022].

G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL http://arxiv.org/abs/1505.00853.

Z. Zhang, Y., J. Zhou, S. Duan, H. Zhao, and R. Wang. Sg-net: Syntax guided transformer for language representation. *CoRR*, abs/2012.13915, 2020. URL https://arxiv.org/abs/2012.13915.

J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2021.01.001. URL https://www.sciencedirect.com/science/article/pii/S2666651021000012.