Do Syntax Trees Help Pre-trained Transformers Extract Information?

Devendra Singh Sachan[1,2], **Yuhao Zhang**[3], **Peng Qi**[3], **William Hamilton**[1,2]
[1]Mila - Quebec AI Institute
[2]School of Computer Science, McGill University
[3]Stanford University
sachande@mila.quebec, wlh@cs.mcgill.ca
{yuhaozhang, pengqi}@stanford.edu

Abstract

Much recent work suggests that incorporating syntax information from dependency trees can improve task-specific transformer models. However, the effect of incorporating dependency tree information into *pre-trained* transformer models (e.g., BERT) remains unclear, especially given recent studies highlighting how these models implicitly encode syntax. In this work, we systematically study the utility of incorporating dependency trees into pre-trained transformers on three representative information extraction tasks: semantic role labeling (SRL), named entity recognition, and relation extraction.

We propose and investigate two distinct strategies for incorporating dependency structure: a *late fusion* approach, which applies a graph neural network on the output of a transformer, and a *joint fusion* approach, which infuses syntax structure into the transformer attention layers. These strategies are representative of prior work, but we introduce additional model design elements that are necessary for obtaining improved performance. Our empirical analysis demonstrates that these syntax-infused transformers obtain state-of-the-art results on SRL and relation extraction tasks. However, our analysis also reveals a critical shortcoming of these models: we find that their performance gains are highly contingent on the availability of human-annotated dependency parses, which raises important questions regarding the viability of syntax-augmented transformers in real-world applications.[1]

1 Introduction

Dependency trees—a form of syntactic representation that encodes an asymmetric syntactic relation between words in a sentence, such as *subject* or *adverbial modifier*—have proven very useful in various NLP tasks. For instance, features defined in terms of the shortest path between entities in a dependency tree were used in relation extraction (RE) (Fundel et al., 2006; Björne et al., 2009), parse structure has improved named entity recognition (NER) (Jie et al., 2017), and joint parsing was shown to benefit semantic role labeling (SRL) (Pradhan et al., 2005) systems. More recently, dependency trees have also led to meaningful performance improvements when incorporated into neural network models for these tasks. Popular encoders to include dependency tree into neural models include graph neural networks (GNNs) for SRL (Marcheggiani and Titov, 2017) and RE (Zhang et al., 2018), and biaffine attention in transformers for SRL (Strubell et al., 2018).

In parallel, there has been a renewed interest in investigating self-supervised learning approaches to pre-training neural models for NLP, with recent successes including ELMo (Peters et al., 2018), GPT (Radford et al., 2018), and BERT (Devlin et al., 2019). Of late, the BERT model based on pre-training of a large transformer model (Vaswani et al., 2017) to encode bidirectional context has emerged as a dominant paradigm, thanks to its improved modeling capacity which has led to state-of-the-art results in many NLP tasks.

BERT's success has also attracted attention to what linguistic information its internal representations capture. For example, Tenney et al. (2019) attribute different linguistic information to different BERT layers; Clark et al. (2019) analyze BERT's attention heads to find syntactic dependencies; Hewitt and Manning (2019) show evidence that BERT's hidden representation embeds syntactic trees. However, it remains unclear if this linguistic information helps BERT in downstream tasks during finetuning or not. Further, it is not evident if external syntactic information can further improve

[1]Our code is available at: https://github.com/DevSinghSachan/syntax-augmented-bert

BERT's performance on downstream tasks.

In this paper, we investigate the extent to which pre-trained transformers can benefit from integrating external dependency tree information. We perform the first systematic investigation of how dependency trees can be incorporated into pre-trained transformer models, focusing on three representative information extraction tasks where dependency trees have been shown to be particularly useful for neural models: semantic role labeling (SRL), named entity recognition (NER), and relation extraction (RE).

We propose two representative approaches to integrate dependency trees into pre-trained transformers (i.e., BERT) using syntax-based graph neural networks (syntax-GNNs). The first approach involves a sequential assembly of a transformer and a syntax-GNN, which we call *Late Fusion*, while the second approach interleaves syntax-GNN embeddings within transformer layers, termed *Joint Fusion*. These approaches are inspired by recent work that combines transformers with external input, but we introduce design elements such as the alignment between dependency tree and BERT wordpieces that lead to obtaining strong performance. Comprehensive experiments using these approaches reveal several important insights:

- Both our syntax-augmented BERT models achieve a new state-of-the-art on the CoNLL-2005 and CoNLL-2012 SRL benchmarks when the gold trees are used, with the best variant outperforming a fine-tuned BERT model by over 3 F_1 points on both datasets. The Late Fusion approach also provides performance improvements on the TACRED relation extraction dataset.
- These performance gains are consistent across different pre-trained transformer approaches of different sizes (i.e. $\text{BERT}_{\text{BASE/LARGE}}$ and $\text{RoBERTa}_{\text{BASE/LARGE}}$).
- The Joint Fusion approach that interleaves GNNs with BERT achieves higher performance improvements on SRL, but it is also less stable and more prone to errors when using noisy dependency tree inputs such as for the RE task, where Late Fusion performs much better, suggesting complementary strengths from both approaches.
- In the SRL task, the performance gains of both approaches are highly contingent on the availability of human-annotated parses for both training and inference, without which the performance gains are either marginal or non-existent. In the NER task, even the gold trees don't show performance improvements.

Although our work does obtain new state-of-the-art results on SRL tasks by introducing dependency tree information from syntax-GNNs into BERT, however, our most important result is somewhat negative and cautionary: the performance gains are only substantial when human-annotated parses are available. Indeed, we find that even high-quality automated parses generated by domain-specific parsers do not suffice, and we are only able to achieve meaningful gains with human-annotated parses. This is a critical finding for future work—especially for SRL—as researchers routinely develop models with human-annotated parses, with the implicit expectation that models will generalize to high-quality automated parses.

Finally, our analysis provides indirect evidence that pre-trained transformers do incorporate sufficient syntactic information to achieve strong performance on downstream tasks. While human-annotated parses can still help greatly, with our proposed models it appears that the knowledge in automatically extracted syntax trees is largely redundant with the implicit syntactic knowledge learned by pre-trained models such as BERT.

2 Models

In this section, we will first briefly review the transformer encoder, then describe the graph neural network (GNN) that learns syntax representations using dependency tree input, which we term the syntax-GNN. Next, we will describe our syntax-augmented BERT models that incorporate such representations learned from the GNN.

2.1 Transformer Encoder

The transformer encoder (Vaswani et al., 2017) consists of three core modules in sequence: embedding layer, multiple encoder layers, and a task-specific output layer. The core elements in these modules are different sets of learnable weight matrices that perform linear transformations. The embedding layer consists of wordpiece embeddings, positional embeddings, and segment embeddings (Devlin et al., 2019). After embedding lookup, these three embeddings are added to obtain token embeddings for an input sentence. The encoder layers then transform the input token embeddings to hidden state representations. The encoder layer con-

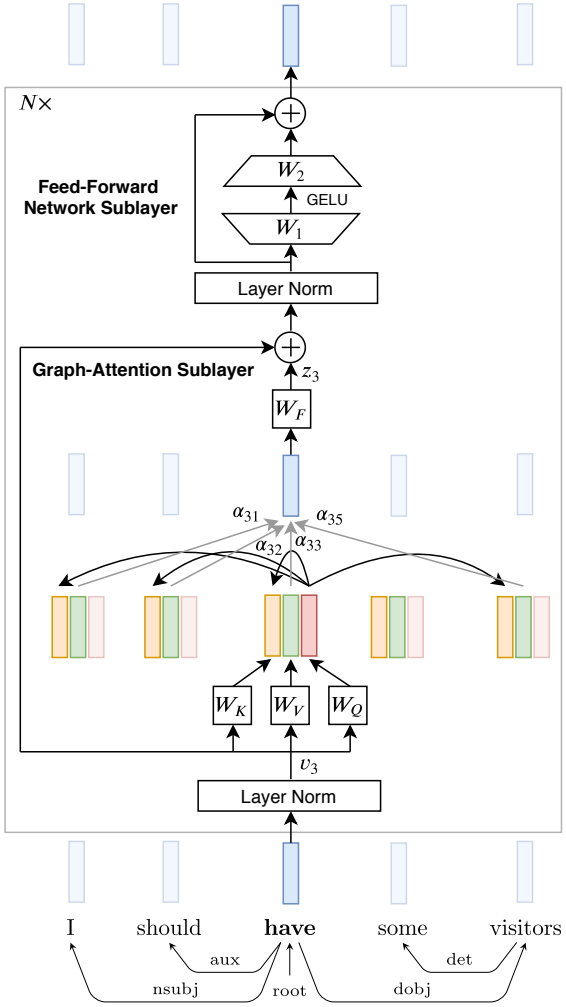


Figure 1: Block diagram illustrating syntax-GNN applied over a sentence's dependency tree. In the example shown, for the word "*have*", the graph-attention sublayer aggregates representations from its three adjacent nodes in the dependency graph.

sists of two sublayers: multi-head dot-product self-attention and feed-forward network, which will be covered in the following section. Finally, the output layer is task-specific and consists of one layer feed-forward network.

2.2 Syntax-GNN: Graph Neural Network over a Dependency Tree

A dependency tree can be considered as a multi-attribute directed graph where the nodes represent words and the edges represent the dependency relation between the head and dependent words. To learn useful syntax representations from the dependency tree structure, we apply graph neural networks (GNNs) (Hamilton et al., 2017; Battaglia et al., 2018) and henceforth call our model *syntax-GNN*. Our syntax-GNN encoder as shown in Figure 1 is a variation of the transformer encoder where the self-attention sublayer is replaced by *graph attention* (Veličković et al., 2018). Self-attention can also be considered as a special case of graph-attention where each word is connected to all the other words in the sentence.

Let $V = \{v_i \in \mathbb{R}^d\}_{i=1:\mathcal{N}^v}$ denote the input node embeddings and $E = \{(e_k, i, j)_{k=1:\mathcal{N}^e}\}$ denote the edges in the dependency tree, where the edge e_k is incident on nodes i and j. Each layer in our syntax-GNN encoder consists of two sublayers: graph attention and feed-forward network.

First, interaction scores (s_{ij}) are computed for all the edges by performing dot-product on the adjacent linearly transformed nodes embeddings

$$s_{ij} = (v_i \boldsymbol{W}_Q)(v_j \boldsymbol{W}_K)^\top. \quad (1)$$

The terms $v_i \boldsymbol{W}_Q$ and $v_i \boldsymbol{W}_K$ are also known as *query* and *key* vectors respectively. Next, an attention score (α_{ij}) is computed for each node by applying softmax over the interaction scores from all its connecting edges:

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(s_{ik})}, \quad (2)$$

where $\mathcal{N}_i$ refers to the set of nodes connected to i^{th} node. The graph attention output (z_i) is computed by the aggregation of attention scores followed by a linear transformation:

$$z_i = \Big(\sum_{j \in \mathcal{N}_i} \alpha_{ij}(v_j \boldsymbol{W}_V)\Big)\boldsymbol{W}_F. \quad (3)$$

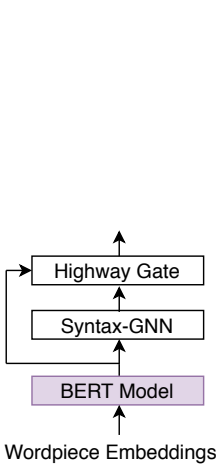
The term $v_j \boldsymbol{W}_V$ is also referred to as *value* vector. Subsequently, the message (z_i) is passed to the second sublayer that consists of two layer fully connected feed-forward network with GELU activation (Hendrycks and Gimpel, 2016).

$$\text{FFN}(z_i) = \text{GELU}(z_i \boldsymbol{W_1} + b_1)\boldsymbol{W_2} + b_2\,. \quad (4)$$

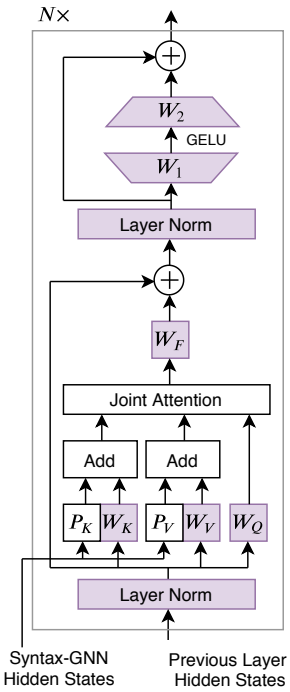
The FFN sublayer outputs are given as input to the next layer. In the above equations $\boldsymbol{W}_K$, $\boldsymbol{W}_V$, $\boldsymbol{W}_Q$, $\boldsymbol{W}_F$, $\boldsymbol{W_1}$, $\boldsymbol{W_2}$ are trainable weight matrices and b_1, b_2 are bias parameters. Additionally, layer normalization (Ba et al., 2016) is applied to the input and residual connections (He et al., 2016) are added to the output of each sublayer.

2.2.1 Dependency Tree over Wordpieces

As BERT models take as input subword units (also known as wordpieces) instead of linguistic tokens,



(a) Late Fusion



(b) Joint Fusion

Figure 2: Block diagrams illustrating our proposed syntax-augmented BERT models. Weights shown in color are pre-trained while those not colored are either non-parameterized operations or have randomly initialized weights. The inputs to each of these models are wordpiece embeddings while their output goes to task-specific output layers. In subfigure 2b, $N\times$ indicates that there are N layers, with each of them being passed the same set of syntax-GNN hidden states.

this also necessitates to extend the definition of a dependency tree to include wordpieces. For this, we introduce additional edges in the original dependency tree by defining new edges from the first subword (head word) of a token to the remaining subwords (tail words) of the same token.

2.3 Syntax-Augmented BERT

In this section, we propose parameter augmentations over the BERT model to best incorporate syntax information from a syntax-GNN. To this end, we introduce two models— Late Fusion and Joint Fusion. These models represent novel mechanisms—inspired by previous work—through which syntax-GNN features are incorporated at different sublayers of BERT (Figure 2). We refer to these models as Syntax-Augmented BERT (SA-BERT) models. During the finetuning step, the new parameters in each model are randomly initialized while the existing parameters are initialized from pre-trained BERT.

Late Fusion: In this model, we feed the BERT contextual representations to the syntax-GNN encoder *i.e.* syntax-GNN is stacked over BERT (Figure 2a). We also use a Highway Gate (Srivastava et al., 2015) at the output of the syntax-GNN encoder to adaptively select useful representations for the training task. Concretely, if v_i and z_i are the representations from BERT and syntax-GNN respectively, then the output (h_i) after the gating layer is computed as,

$$g_i = \sigma(\boldsymbol{W}_g v_i + b_g) \tag{5}$$

$$h_i = g_i \odot v_i + (1 - g_i) \odot z_i, \tag{6}$$

where σ is the sigmoid function $1/\left(1 + e^{-x}\right)$ and $\boldsymbol{W}_g$ is a learnable parameter. Finally, we map the output representations to linguistic space by adding the hidden states of all the wordpieces that map to the same linguistic token respectively.

Joint Fusion: In this model, syntax representations are incorporated within the self-attention sublayer of BERT. The motivation is to *jointly* attend over both syntax- and BERT representations. First, the syntax-GNN representations are computed from the input token embeddings and its final layer hidden states are passed to BERT. Second, as is shown in Figure 2b, the syntax-GNN hidden states are linearly transformed using weights $\boldsymbol{P}_K$, $\boldsymbol{P}_V$ to obtain additional *syntax-based* key and value vectors. Third, syntax-based key and value vectors are added with the BERT's self-attention sublayer key and value vectors respectively. Fourth, the query vector in self-attention layer now attends over this set of keys and values, thereby augmenting the model's ability to fuse syntax information. Overall, in this model, we introduce two new set of weights per layer $\{\boldsymbol{P}_K, \boldsymbol{P}_V\}$, which are randomly initialized.

3 Experimental Setup

3.1 Tasks and Datasets

For our experiments, we consider information extraction tasks for which dependency trees have been extensively used in the past to improve model performance. Below, we provide a brief description of these tasks and the datasets used and refer the reader to Appendix A.1 for full details.

Semantic Role Labeling (SRL) In this task, the objective is to assign semantic role labels to text spans in a sentence such that they answer the query:

Who did *what* to *whom* and *when*? Specifically, for every target *predicate* (verb) of a sentence, we detect *syntactic constituents* (arguments) and classify them into predefined semantic roles. In our experiments, we study the setting where the predicates are given and the task is to predict the arguments. We use the CoNLL-2005 SRL corpus (Carreras and Màrquez, 2005) and CoNLL-2012 OntoNotes[2] dataset, which contains PropBank-style annotations for predicates and their arguments, and also includes POS tags and constituency parses.

Named Entity Recognition (NER) NER is the task of recognizing entity mentions in text and tagging them to entity categories. We use the OntoNotes 5.0 dataset (Pradhan et al., 2012), which contains 18 named entity types.

Relation Extraction (RE) RE is the task of predicting the relation between the two entity mentions in a sentence. We use the label corrected version of the TACRED dataset (Zhang et al., 2017; Alt et al., 2020), which contains 41 relation types as well as a special *no_relation* class indicating that no relation exists between the two entities.

3.2 Training Details

We select *bert-base-cased* to be our reference pre-trained baseline model.[3] It consists of 12 layers, 12 attention heads, and 768 model dimensions. In both the variants, the syntax-GNN component consists of 4 layers, while other configurations are kept the same as *bert-base*. Also, for the Joint Fusion method, syntax-GNN hidden states were shared across different layers. It is worth noting that as our objective is to assess if the use of dependency trees can provide performance gains over pre-trained transformer models, it is important to tune the hyperparameters of these baseline models to obtain strong reference scores. Therefore, for each task, during the finetuning step, we tune the hyperparameters of the default *bert-base* model and use the same hyperparameters to train the SA-BERT models. We refer the reader to Appendix A.2 for additional training details.

4 Results and Analysis

In this section, we present our main empirical analyses and key findings.

[2] conll.cemantix.org/2012/data.html

[3] *bert-base* configuration was preferred due to computational reasons and we found that *bert-cased* provided substantial gains over *bert-uncased* in the NER task.

Test Set	P	R	F_1
Baseline Models (without dependency parses)			
SA+GloVe†	84.17	83.28	83.72
SA+ELMo†	86.21	85.98	86.09
$\text{BERT}_{\text{BASE}}$	86.97	88.01	87.48
Gold Dependency Parses			
Late Fusion	89.17	91.09	90.12
Joint Fusion	90.59	91.35	**90.97**

Table 1: SRL results on the CoNLL-2005 WSJ test set averaged over 5 independent runs. † marks results from Strubell et al. (2018).

Test Set	P	R	F_1
Baseline Models (without dependency parses)			
SA+GloVe†	82.55	80.02	81.26
SA+ELMo†	84.39	82.21	83.28
Deep-LSTM+ELMo‡	-	-	84.60
Structure-distilled BERT*	-	-	86.39
$\text{BERT}_{\text{BASE}}$	85.91	87.07	86.49
Gold Dependency Parses			
Late Fusion	88.06	90.32	89.18
Joint Fusion	89.34	90.44	**89.89**

Table 2: SRL results on the CoNLL-2012 test set averaged over 5 independent runs. † marks results from Strubell et al. (2018); ‡ mark result from Peters et al. (2018); ∗ mark result from Kuncoro et al. (2020).

4.1 Benchmark Performance

To recap, our two proposed variants of the Syntax-Augmented BERT models in Section 2.3 mainly differ at the position where syntax-GNN outputs are fused with the BERT hidden states. Following this, we first compare the effectiveness of these variants on all the three tasks, comparing against previous state-of-the-art systems such as (Strubell et al., 2018; Jie and Lu, 2019; Zhang et al., 2018), which are outlined in Appendix B due to space limitations. For this part we use gold dependency parses to train the models for SRL and NER, and predicted parses for RE, since gold dependency parses are not available for TACRED.

We present our main results for SRL in Table 1 and Table 2, NER in Table 3, and RE in Table 4. All these results report average performance over five runs with different random seeds. First, we note that for all the tasks, our *bert-base* baseline is quite strong and is directly competitive with other state-of-the-art models.

We observe that both the Late Fusion and Joint Fusion variants of our approach yielded the best results in the SRL tasks. Specifically, on CoNLL-

Test Set	P	R	F_1
Baseline Models (without dependency parses)			
BiLSTM-CRF+ELMo†	88.25	89.71	88.98
$BERT_{BASE}$	88.75	89.61	89.18
Gold Dependency Parses			
DGLSTM-CRF+ELMo†	89.59	90.17	**89.88**
Late Fusion	88.75	89.19	88.97
Joint Fusion	88.58	89.31	88.94

Table 3: NER results on the OntoNotes-5.0 test set averaged over 5 independent runs. † marks results from Jie and Lu (2019).

Test Set	P	R	F_1
Baseline Models (without dependency parses)			
$BERT_{BASE}$	78.04	76.36	77.09
Stanford CoreNLP Dependency Parses			
GCN†	74.2	69.3	71.7
GCN+$BERT_{BASE}$†	74.8	74.1	74.5
Late Fusion	78.55	76.29	**77.38**
Joint Fusion	70.22	75.12	72.52

Table 4: Relation extraction results on the revised TACRED test set (Alt et al., 2020), averaged over 5 independent runs. † marks results reported by Alt et al. (2020).

2005 and CoNLL-2012 SRL, Joint Fusion improves over *bert-base* by an absolute 3.5 F_1 points, while Late Fusion improves over *bert-base* by 2.65 F_1 points. On the RE task, the Late Fusion model improves over *bert-base* by approximately 0.3 F_1 points while the Joint Fusion model leads to a drop of 4.5 F_1 points in performance (which we suspect is driven by the longer sentence lengths observed in TACRED). On NER, the SA-BERT models lead to no performance improvements as their scores lies within one standard deviation to that of *bert-base*.

Overall, we find that syntax information is most useful to the pre-trained transformer models in the SRL task, especially when intermixing the intermediate representations of BERT with representations from the syntax-GNN. Moreover, when the fusion is done after the final hidden layer of the pre-trained models, apart from providing good gains on SRL, it also provides small gains on RE task. We further note that, as we trained all our syntax-augmented BERT models using the same hyperparameters as that of *bert-base*, it is possible that separate hyperparameter tuning would further improve their performance.

4.2 Impact of Parsing Quality

In this part, we study to what extent parsing quality can affect the performance results of the syntax-augmented BERT models. Specifically, following existing work, we compare the effect of using parse trees from three different sources: (a) gold syntactic annotations[4]; (b) a dependency parser trained using gold, *in-domain* parses[5]; and (c) available off-the-shelf NLP toolkits.[6] In previous work, it was shown that using in-domain parsers can provide good improvements on SRL (Strubell et al., 2018) and NER tasks (Jie and Lu, 2019), and the performance can be further improved when gold parses were used at test time. Meanwhile, in many practical settings where gold parses are not readily available, the only option is to use parse trees produced by existing NLP toolkits, as was done by Zhang et al. (2018) for RE. In these cases, since the parsers are trained on a different domain of text, it is unclear if the produced trees, when used with the SA-BERT models, can still lead to performance gains. Motivated by these observations, we investigate to what extent *gold*, *in-domain*, and *off-the-shelf* parses can improve performance over strong BERT baselines.

Comparing off-the-shelf and gold parses. We report our findings on the CoNLL-2005 SRL (Table 5), CoNLL-2012 SRL (Table 6), and OntoNotes-5.0 NER (Table 7) tasks. Using gold parses, both the Late Fusion and Joint Fusion models obtain greater than 2.5 F_1 improvement on SRL compared with *bert-base* while we don't observe significant improvements on NER. We further note that as the gold parses are produced by expert human annotators, these results can be considered as the attainable performance ceiling from using parse trees in these models.

We also observe that using off-the-shelf parses from the Stanza toolkit (Qi et al., 2020) provides little to no gains in F_1 scores (see Tables 5 and 7). This is mainly due to the low in-domain accuracy of the predicted parses. For example, on the CoNLL-

[4]We use Stanford *head rules* (de Marneffe and Manning, 2008) implemented in Stanford CoreNLP v4.0.0 (Manning et al., 2014) to convert constituency trees to dependency trees in UDv2 format (Nivre et al., 2020).

[5]The difference between settings (a) and (b) is during test time. In (a) gold parses are used for both training and test instances while in (b) gold parses are used for training, while during test time, parses are extracted from a dependency parser which was trained using gold parses.

[6]In this setting, the parsers are trained on general datasets such as the Penn Treebank or the English Web Treebank.

Test Set	P	R	F_1
Stanza Dependency Parses (UAS: 84.20)			
Late Fusion	86.85	88.06	87.45
Joint Fusion	86.87	87.85	87.36
In-domain Dependency Parses (UAS: 92.66)			
LISA+GloVe†	85.53	84.45	84.99
LISA+ELMo†	87.13	86.67	86.90
Late Fusion	86.80	87.98	87.39
Joint Fusion	87.09	87.95	87.52
Gold Dependency Parses			
Late Fusion	89.17	91.09	90.12
Joint Fusion	90.59	91.35	**90.97**

Table 5: SRL results with different parses on the CoNLL-2005 WSJ test set averaged over 5 independent runs. † marks results from Strubell et al. (2018).

2012 SRL test set the UAS is 84.2% for the Stanza parser, which is understandable as the parser was trained on the EWT corpus which covers a different domain.

In a more fine-grained error analysis, we also examined the correlation between parse quality and performance on individual examples on CoNLL-2005 (Figures 3a and 3b), finding a mild but significant positive correlation between parse quality and relative model performance when training and testing with Stanza parses (Figure 3a). Interestingly, we found that this correlation between parse quality and validation performance is much stronger when we train a model on gold parses but then evaluate with noisy Stanza parses (Figure 3b). This suggests that the model trained on noisy parses tends to rely less on the noisy dependency tree inputs, while the model trained on gold parses is more sensitive to the external syntactic input. This correlation is further reinforced by our manual error analysis presented in Appendix C (Figures 4 and 5), where we show how the erroneous edges in the Stanza parses can lead to incorrect predictions of the SRL tags.

Do in-domain parses help? Lastly, for the setting of using in-domain parses, we only evaluate on the SRL task, since on the NER task even using gold parses does not yield substantial gain. We train a biaffine parser (Dozat and Manning, 2017) on the gold parses from the CoNLL-2005 training set and obtain parse trees from it at test time. We observe that while the obtained parse trees are fairly accurate (with a UAS of 92.6% on the test set), it leads to marginal or no improvements over *bert-base*. This finding is also similar to the results obtained by Strubell et al. (2018), where their

Test Set	P	R	F_1
Stanza Dependency Parses (UAS: 82.73)			
Late Fusion	85.74	87.18	86.45
Joint Fusion	85.94	87.05	86.49
In-domain Dependency Parses (UAS: 93.60)			
Late Fusion	86.06	86.90	86.48
Joint Fusion	85.75	86.92	86.33
Gold Dependency Parses			
Late Fusion	88.06	90.32	89.18
Joint Fusion	89.34	90.44	**89.89**

Table 6: SRL results with different parses on the CoNLL-2012 test set.

Test Set	P	R	F_1
Stanza Dependency Parses (UAS: 83.91)			
Late Fusion	88.83	89.42	89.12
Joint Fusion	88.56	89.38	88.97
In-domain Dependency Parses (UAS: 96.10)			
DGLSTM-CRF+ELMo†	–	–	89.64
Gold Dependency Parses			
DGLSTM-CRF+ELMo†	89.59	90.17	**89.88**
Late Fusion	88.75	89.19	88.97
Joint Fusion	88.58	89.31	88.94

Table 7: NER results with different parses on the OntoNotes-5.0 test set averaged over 5 independent runs. † marks results from Jie and Lu (2019).

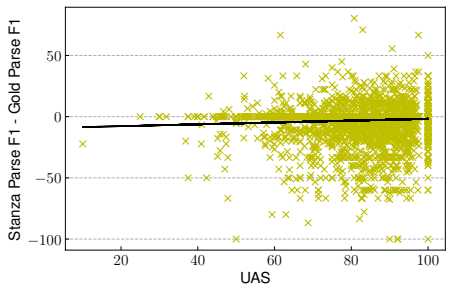
LISA+ELMo model only obtains a relatively small improvement over SA+ELMo. We hypothesize that as the accuracy of the predicted parses further increases, the F_1 scores would be closer to that from using the gold parses. One possible reason for these marginal gains from using the in-domain parses is that as they are still imperfect, the errors in the parse edges is forcing the model to ignore the syntax information.

Overall, we conclude that *parsing quality has a drastic impact on the performance of the Syntax-Augmented BERT models, with substantial gains only observed when gold parses are used.*

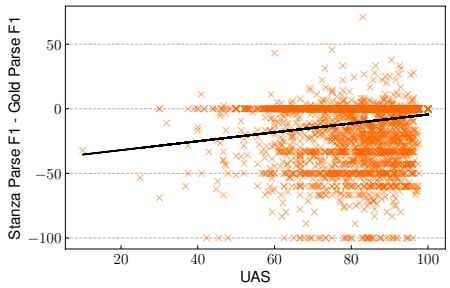
5 Generalizing to BERT Variants

Our previous results used the *bert-base* setting, which is a relatively small configuration among pre-trained models. Devlin et al. (2019) also proposed larger model settings (*bert-large*[7], *whole-word-masking*[8]) that outperformed *bert-base* in all the benchmark tasks. More recently, Liu et al. (2019)

[7] 24 layers, 16 attention heads, 1024 model dimensions
[8] https://bit.ly/3l7rbXx



(a) When models are trained using Stanza and gold parses, we observe a small positive correlation between F_1 difference and UAS, suggesting that as UAS of Stanza parse increases, the model makes less errors. The slope of the fitted linear regression model is 0.075 and the intercept is -9.27.



(b) Inference is done using Stanza parses on a model trained with gold parses. The slope of the fitted linear regression model is 0.345 and the intercept is -38.9.

Figure 3: Correlation between parse quality and difference in F_1 scores on CoNLL-2005 SRL WSJ dataset.

proposed RoBERTa, a better-optimized variant of BERT that demonstrated improved results. A research question that naturally arises is: *Is syntactic information equally useful for these more powerful pre-trained transformers, which were pre-trained in a different way than bert-base?* To answer this, we finetune these models—with and without Late Fusion—on the CoNLL-2005 SRL task using gold parses and report their performance in Table 8.[9]

As expected, we observe that both *bert-large* and *bert-wwm* models outperform *bert-base*, likely due to the larger model capacity from increased width and more layers. Our Late Fusion model consistently improves the results over the underlying BERT models by about 2.2 F_1. The RoBERTa models achieve improved results compared with the BERT models. And again, our Late Fusion model further improves the RoBERTa results by about 2 F_1. Thus, it is evident that *the gains from the Late Fusion model generalize to other widely used pre-trained transformer models*.

[9]We use the *Late Fusion* model with *gold parses* in this section, as it is computationally more efficient to train than Joint Fusion model.

Gold Parses	P	R	F_1
	BERT		
$BERT_{LARGE}$	88.14	88.84	88.49
Late Fusion	89.86	91.57	**90.70**
$BERT_{WWM}$	88.04	88.87	88.45
Late Fusion	89.88	91.63	**90.75**
	RoBERTa		
$RoBERTa_{LARGE}$	89.14	89.90	89.47
Late Fusion	90.89	92.08	**91.48**

Table 8: SRL results from using different pre-trained models on the CoNLL-2005 WSJ test set averaged over 5 independent runs. WWM indicates the whole-wordpiece-masking.

6 Generalizing to Out-of-Domain Data

In real-world applications, NLP systems are often used in a domain different from training. And it was previously shown that many NLP systems, such as information extraction systems, suffer from substantial performance degradation when applied to out-of-domain data (Huang and Yates, 2010). While it is evident that syntax trees may help models generalize to out-of-domain data (Wang et al., 2017), since the inductive biases introduced by these trees are invariant across domains, it is unclear if this hypothesis holds for more recent pre-trained models. To study this, we run experiments on SRL with the CoNLL-2005 SRL corpus because this is where we have access to both in-domain and out-of-domain test data using the same annotation schema. The training set of this corpus contains WSJ articles from the newswire domain and the test set consists of two splits: WSJ articles (in-domain) and Brown corpus[10] (out-of-domain). For training, we use both BERT and RoBERTa pre-trained models and leverage gold parses in syntax-GNN models.

From the results in Table 9, the utility of syntax-GNN is evident, as we find that the Late Fusion model always improves over its corresponding BERT and RoBERTa baselines by 2-3% relative F_1, with RoBERTa-large based Late Fusion achieving the best F_1 on both WSJ and Brown datasets. We also compare the performance across both domains, with the last column showing the relative drop in the F_1 score between WSJ and Brown datasets. We observe that the performance of all models drops substantially on the Brown set. However, compared with randomly initialized transformer

[10]contains text from 15 genres (Francis and Kucera, 1979)

	WSJ Test		*Brown Test*		
Gold Parses	**F$_1$**	% Δ	**F$_1$**	% Δ	% ∇
		Baseline Models			
SA+GloVe†	84.5		73.1		13.5
LISA+GloVe†	86.0	1.8	76.5	4.7	11.0
		BERT			
BERT$_{\text{BASE}}$	87.5		81.5		6.9
Late Fusion	**90.1**	3.0	**83.9**	2.9	6.9
BERT$_{\text{LARGE}}$	88.5		82.5		6.8
Late Fusion	**90.8**	2.6	**84.6**	2.5	6.8
		RoBERTa			
RoBERTa$_{\text{LARGE}}$	89.5		84.0		6.1
Late Fusion	**91.5**	2.2	**85.5**	1.8	6.6

Table 9: Out-of-domain SRL results on the CoNLL-2005 WSJ and Brown test sets. † marks results reported in Strubell et al. (2018). %Δ denotes the relative gain in F$_1$ over pre-trained models when using Late Fusion model, %∇ denotes the relative drop in F$_1$ when a model trained on WSJ dataset is tested on the Brown dataset.

models, where the results can drop by 13%, both syntax-fused and pre-trained models lead to better generalization as the relative error drop reduces to 6–7%. *We see that using Late Fusion does not lead to a better out-of-domain generalization, when compared to strong pre-trained transformers without using parse trees.* Lastly, we find that among all pre-trained models, RoBERTa-large and its syntax-fused variant Late Fusion achieves the lowest out-of-domain generalization error.

7 Related Work

Our work is based on finetuning large pre-trained transformer models for NLP tasks, and is closely related to existing work on understanding the syntactic information encoded in them, which we have earlier covered in Section 1. Here we instead focus on discussing related work that studies incorporating syntax into neural NLP models.

Relation Extraction Neural network models have shown performance improvements when shortest dependency path between entities was incorporated in sentence encoders: Liu et al. (2015) apply a combination of recursive neural networks and CNNs; Miwa and Bansal (2016) apply tree-LSTMs for joint entity and relation extraction; and Zhang et al. (2018) apply graph convolutional networks (GCN) over LSTM features.

Semantic Role Labeling Recently, several approaches have been proposed to incorporate dependency trees within neural SRL models such as learning the embeddings of dependency path between predicate and argument words (Roth and Lapata, 2016); combining GCN-based dependency tree representations with LSTM-based word representations (Marcheggiani and Titov, 2017); and linguistically-informed self-attention in one transformer attention head (Strubell et al., 2018). Kuncoro et al. (2020) directly inject syntax information into BERT pre-training through knowledge distillation, an approach which improves the performance on several NLP tasks including SRL.

Named Entity Recognition Moreover, syntax has also been found to be useful for NER as it simplifies modeling interactions between multiple entity mentions in a sentence (Finkel and Manning, 2009). To model syntax on OntoNotes-5.0 NER task, Jie and Lu (2019) feed the concatenated child token, head token, and relation embeddings to LSTM and then fuse child and head hidden states.

8 Conclusion

In this work, we explore the utility of incorporating syntax information from dependency trees into pre-trained transformers when applied to information extraction tasks of SRL, NER, and RE. To do so, we compute dependency tree embeddings using a syntax-GNN and propose two models to fuse these embeddings into transformers. Our experiments reveal several important findings: syntax representations are most helpful for SRL task when fused within the pre-trained representations, these performance gains on SRL task are contingent on the quality of the dependency parses. We also notice that these models don't provide any performance improvements on NER. Lastly, for the RE task, syntax representations are most helpful when incorporated on top of pre-trained representations.

Acknowledgements

The authors would like to thank Mrinmaya Sachan, Xuezhe Ma, Siva Reddy, and Xavier Carreras for providing us valuable feedback that helped to improve the paper. We would also like to thank the anonymous reviewers for giving us their useful suggestions about this work. This project was supported by academic gift grants from IBM and Microsoft Research, as well as a Canada CIFAR AI Chair held by Prof. Hamilton.

References

Christoph Alt, Aleksandra Gabryszak, and Leonhard Hennig. 2020. TACRED revisited: A thorough evaluation of the TACRED relation extraction task. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Jari Björne, Juho Heimonen, Filip Ginter, Antti Airola, Tapio Pahikkala, and Tapio Salakoski. 2009. Extracting complex biological events with rich graph-based feature sets. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? An analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers*.

Timothy Dozat and Christopher D Manning. 2017. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations (ICLR)*.

Jenny Rose Finkel and Christopher D. Manning. 2009. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

G. D. Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

W. N. Francis and H. Kucera. 1979. Brown corpus manual. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US.

Katrin Fundel, Robert Küffner, and Ralf Zimmer. 2006. RelEx – Relation extraction using dependency parse trees. *Bioinformatics*.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40:52–74.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint arXiv:1606.08415*.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*.

Fei Huang and Alexander Yates. 2010. Open-domain semantic role labeling by modeling word spans. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

Zhanming Jie and Wei Lu. 2019. Dependency-guided LSTM-CRF for named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Zhanming Jie, Aldrian Obaja Muis, and Wei Lu. 2017. Efficient dependency-guided named entity recognition. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *The 2015 International Conference for Learning Representations*.

Adhiguna Kuncoro, Lingpeng Kong, Daniel Fried, Dani Yogatama, Laura Rimell, Chris Dyer, and Phil Blunsom. 2020. Syntactic structure distillation pretraining for bidirectional encoders. *Transactions of the Association for Computational Linguistics*, 8:776–794.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*.

Yang Liu, Furu Wei, Sujian Li, Heng Ji, Ming Zhou, and Houfeng Wang. 2015. A dependency-based neural network for relation classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *COLING 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using LSTMs on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of The 12th Language Resources and Evaluation Conference*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*.

Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James Martin, and Daniel Jurafsky. 2005. Semantic role labeling using different syntactic views. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *Technical Report*.

Michael Roth and Mirella Lapata. 2016. Neural semantic role labeling with dependency path embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. In *Advances in neural information processing systems*.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations (ICLR)*.

Liangguo Wang, Jing Jiang, Hai Leong Chieu, Chen Hui Ong, Dandan Song, and Lejian Liao. 2017. Can syntax help? improving an LSTM-based sentence compression model for new domains. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1385–1393.

Yuhao Zhang, Peng Qi, and Christopher D. Manning. 2018. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

A Experimental Setup

A.1 Task-Specific Modeling Details

Semantic Role Labeling (SRL): We model SRL as a sequence tagging task using a linear-chain CRF (Lafferty et al., 2001) as the last layer. During inference, we perform decoding using the Viterbi algorithm (Forney, 1973). To highlight predicate position in the sentence, we use indicator embeddings as input to the model.

Named Entity Recognition (NER): Similar to SRL, we model NER as a sequence tagging task, and use a linear-chain CRF layer over the model's hidden states. Sequence decoding is performed using the Viterbi algorithm.

Relation Extraction (RE): As is common in prior work (Zhang et al., 2018; Miwa and Bansal, 2016), the dependency tree is pruned such that the subtree rooted at the lowest common ancestor of entity mentions is given as input to the syntax-GNN. Following Zhang et al. (2018), we extract sentence representations by applying a max-pooling operation over the hidden states. We also concatenate the entity representations with sentence representation before the final classification layer.

A.2 Additional Training Details

During the finetuning step, the new parameters in each model are randomly initialized while the existing parameters are initialized from pre-trained BERT. For regularisation, we apply dropout (Srivastava et al., 2014) with $p = 0.1$ to attention coefficients and hidden states. For all datasets, we use the canonical training, development, and test splits. We use the Adam optimizer (Kingma and Ba, 2015) for finetuning.

We observed that the initial learning rate of 2e-5 with a linear decay worked well for all the tasks. For the model training to converge, we found that 10 epochs were sufficient for CoNLL-2012 SRL and RE and 20 epochs were sufficient for CoNLL-2005 SRL and NER. We evaluate the test set performance using the best-performing checkpoint on the development set.

For evaluation, following convention we report the micro-averaged precision, recall, and F_1 scores in every task. For variance control in all the experiments, we report the mean of the results obtained from five independent runs with different seeds.

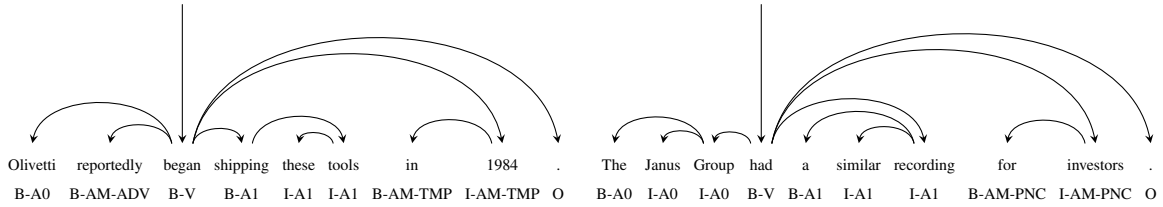
B Additional Baselines

Besides BERT models, we also compare our results to the following previous work, which had obtained good performance gains on incorporating dependency trees with neural models:

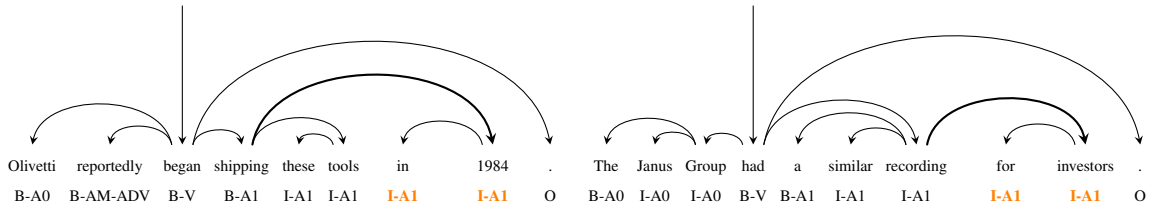
- For SRL, we include results from the SA (self-attention) and LISA (linguistically-informed self-attention) model by Strubell et al. (2018). In LISA, the attention computation in one attention-head of the transformer is biased to enforce dependent words only attend to their head words. The models were trained using both GloVe (Pennington et al., 2014) and ELMo embeddings.
- For NER, we report the results from Jie and Lu (2019), where they concatenate the child token, head token, and relation embeddings as input to an LSTM and then fuse child and head hidden states.
- For RE, we report the results of the GCN model from Zhang et al. (2018) where they apply graph convolutional networks on pruned dependency trees over LSTM states.

C Manual Error Analysis

In this section, we present several examples from our manual error analysis of the predictions from the Late Fusion model when it is trained on CoNLL-2005 SRL WSJ dataset using gold and Stanza parses. Specifically, we show how the incorrect edges present in the parse tree can induce wrong SRL tag predictions. In Figure 4, we observe two examples where the model when trained with gold parses outputs perfect predictions but the when trained with Stanza parses outputs two incorrect SRL tags due to one erroneous edge present in the dependency parse. In Figure 5, we show an example of a longer sentence where due to the presence of four erroneous edges in the Stanza parse, the model makes a series of incorrect predictions of the SRL tags.



(a) Predicted SRL tags using **Gold parses**



(b) Predicted SRL tags using **Stanza parses**

Figure 4: Examples of sentences with their predicted SRL tags when the Late Fusion model is trained using gold parses (4a) and Stanza parses (4b). While the predicted SRL tags using the gold parses are accurate, the erroneous edges in the Stanza parses (highlighted in bold) leads to incorrect SRL tags predictions (highlighted in orange color).

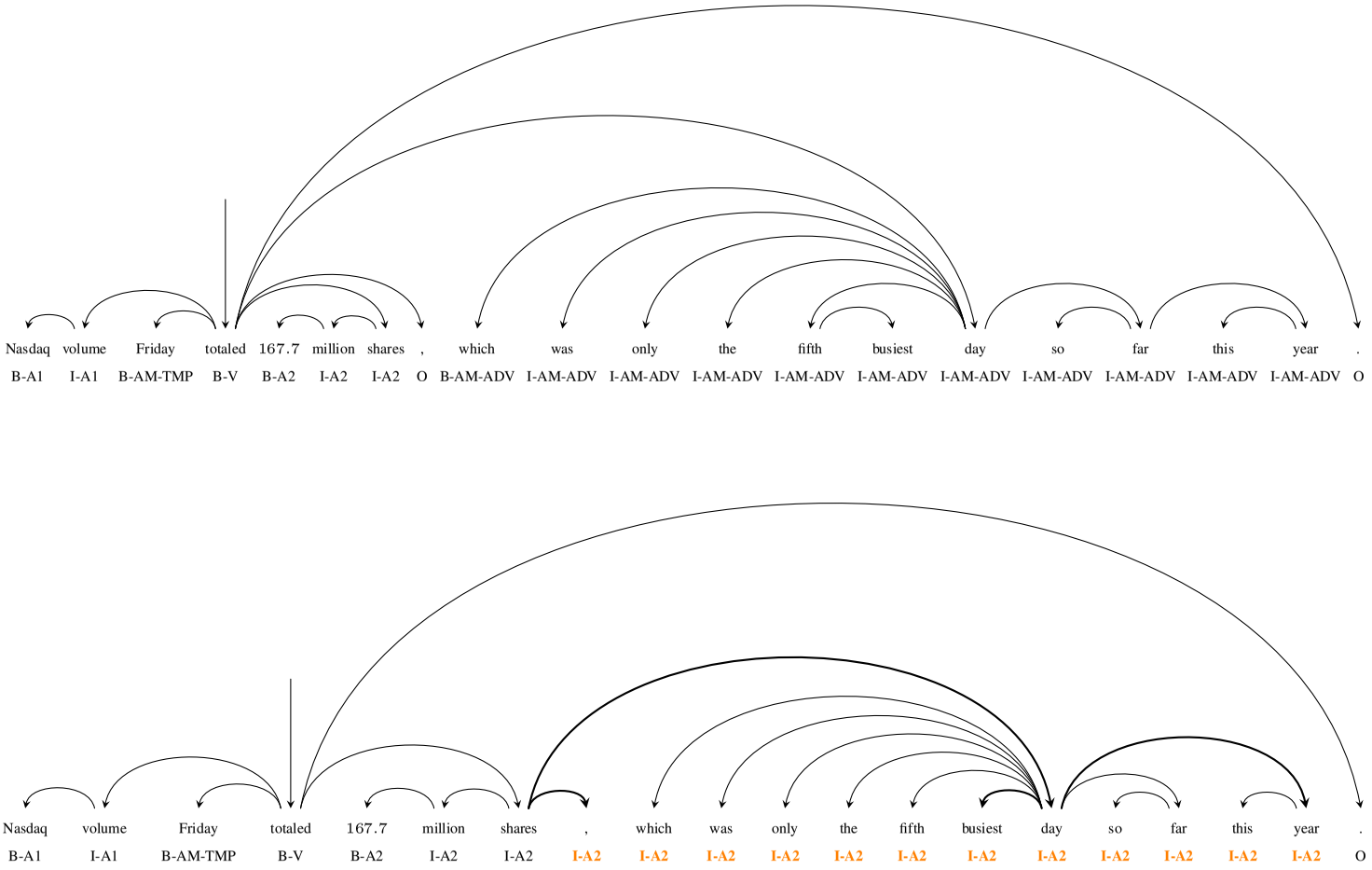


Figure 5: Example of a longer sentence with its predicted SRL tags when the Late Fusion model is trained using gold parses (above figure) and Stanza parses (lower figure). The erroneous edges in the Stanza parses are highlighted in bold. While the predicted SRL tags using the gold parses are accurate, the erroneous edges in the Stanza parses leads to a series of incorrect SRL tag predictions (highlighted in orange color).